TURING

WILEY

谨以此书纪念图灵诞辰百年

图灵的秘密

他的生平、思想及论文解读

The Annotated Turing

【美】Charles Petzold 著

杨卫东 朱皓 等 译



人民邮电出版社

POSTS & TELECOM PRESS

在数字计算机出现之前，阿兰•图灵就预想了它们的功能和通用性……也证明了哪些事是计算机永远做不了的。

由Windows编程大师Charles Petzold耗时多年编写的这本书剖析了现代计算机原理开山之作、阿兰•图灵流芳百世的论文 “On Computable Numbers, with an Application to the Entscheidungs problem”。 图灵在其中描述了一种假想的计算机器，探索了其功能和内在的局限性，由此建立了现代程序设计和可计算性的基础。这本书也像是一本小说，行文间穿插讲述了图灵的成长经历和教育背景， 以及他跌宕起伏的一生，包括破解德国恩尼格密码的传奇经历，他对人工智能的探索，他的性取向，以及最终因同性恋的罪名而在41岁时自杀的悲惨结局。全书完整揭示了阿兰•图灵非凡、传奇而悲剧的一生，是了解图灵的思想和生平的极好著作。

阿兰•图灵（**1912-1954)**

英国数学家、逻辑学家，被称为计算机科学之父、人工智能之父，是计算机逻辑的奠基者，提出了“图灵机”和“图灵测试”等重要概念。为纪念他在计算机领域的卓越贡献，美国计算机协会于1966年设立图灵奖，此奖项被誉为计算机科学界的诺贝尔奖。

Charles Petzold

Windows编程大师、世界顶级技术作家、微软资深MVP，拥有25年的Windows编程经验。1994年5月，Petzold作为唯一的作家，获得由微软公司和*Window Magazine*授予的Windows先锋奖（仅7人获奖），知道今天，他依然是Windows GDI程序设计首席技术作家。他出版过十几本著作，其中包括Win32 API编程经典《Windows程序设计》、《编码》等。

历届图灵奖得主名单

◆ 1966 A. J. Perlis
高级编程技术和编译器架构
◆ 1967 Maurice V. Wilkes
设计出第一台具有内置存储程序的计算机EDSAC
◆ 1968 Richard W. Hamming
数值方法、自动编码系统、错误检测及错误校验码
◆ 1969 Marvin Minsky
创造、推进和提升人工智能
◆ 1970 J. H. Wilkinson
利用数值分析方法来促进高速数字计算机的应用
◆ 1971 John McCarthy
人工智能
◆ 1972 Edsger W. Dijkstra
编程语言
◆ 1973 Charles W. Bachman
数据库
◆ 1974 Donald E. Knuth
算法分析和程序设计语言，“计算机程序设计艺术”丛书
◆ 1975 Allen Newell和Herbert A. Simon
人工智能、人类认知心理学和表处理
◆ 1976 Michael O. Rabin和Dana S. Scott
非确定性机器
◆ 1977 John Backus
可用的高级编程系统设计
◆ 1978 Robert W. Floyd
软件编程的算法，语法分析理论、编程语言的语义和算法分析等多项计算机子学科的创立
◆ 1979 Kenneth E. Iverson
程序设计语言理论、交互系统及APL
◆ 1980 C. Antony R. Hoare

编程语言的定义和设计
◆ 1981 Edgar F. Codd
数据库管理系统的理论和实践
◆ 1982 Stephen A. Cook
奠定了NP完全性理论的基础
◆ 1983 Dennis M. Ritchie和Kenneth L. Thompson
一般操作系统理论，对UNIX操作系统的推广
◆ 1984 Niklaus E. Wirth
开发了EULER、ALGOL-W、MODULA和PASCAL等一系列崭新的计算机语言
◆ 1985 Richard M. Karp
算法理论
◆ 1986 John E. Hopcroft和Robert E. Tarjan
在算法及数据结构的设计和分析中取得了决定性成果
◆ 1987 John Cocke
编译器的理论和设计，大系统体系结构，精简指令集计算机的开发
◆ 1988 Ivan E. Sutherland
计算机图形学
◆ 1989 William V. Kahan
数值分析
◆ 1990 Fernando J. Corbato
组织通用、大规模、分时和资源共享的兼容分时系统和Multics的开发
◆ 1991 Robin Milner
可计算函数逻辑（LCF)、ML和并行理论（CCS)
◆ 1992 Butler W. Lampson
分布式个人计算机系统
◆ 1993 Jurlis Hartmanis和Richard E. Stearns
奠定了计算复杂性理论的基础
◆ 1994 Raj Reddy和Edward Feigenbaum
对大型人工智能系统的开拓性研究
◆ 1995 Manuel Blum
奠定了计算复杂性理论的基础，密码术及程序校验
◆ 1996 Amir Pnueli
在计算中引入时序逻辑、程序及系统检验
◆ 1997 Douglas Engelbart
提出交互计算概念并创造出实现这一概念的重要技术

◆; 1998 James Gray
数据库和事务处理
◆ 1999 Frederick P. Brooks, Jr.
计算机体系结构、操作系统、软件工程
◆ 2000 姚期智（Andrew Chi-Chih Yao)
计算理论方面的基础性工作
◆ 2001 Ole-Johan Dahl和Kristen Nygaard
面向对象程序设计思想
◆ 2002 Ronald L. Rivest、Adi Shamir和Leonard M. Adelman
公共密钥算法（RSA)
◆ 2003 Alan Kay
发明第一个完全面向对象的动态计算机程序设计语言Smalltalk
◆ 2004 Vinton G. Cerf和Robert E. Kahn
在互联网方面的开创性工作
◆ 2005 Peter Naur
Algol 60语言
◆ 2006 Frances E. Allen
编译器优化理论和实践（她是图灵奖第一位女性得主)
◆ 2007 Edmund M. Clarke、Allen Emerson和Joseph Sifakis
将模型校验推广成软硬件工业中广泛采用的高效校验技术
◆ 2008 Barbara Liskov
编程语言和系统设计的实践与理论基础
◆ 2009 Charles P. Thacker
第一台现代个人计算机Alto之父
◆ 2010 Leslie L. Valiant
人工智能、自然语言处理和手写识别等大量革新技术
◆ 2011 Judea Pearl
通过或然性积分和随机推理对人工智能做出贡献

图书在版编目（**CIP**）数据

图灵的秘密：他的生平、思想及论文解读 / （美）佩措尔德（Petzold, C.） 著；杨卫东等译. — 北京 ：人民邮电出版社， 2012. 11
书名原文：The Annotated Turing
ISBN 978-7-115-28214-9

I. ①图… II. ①佩… ②杨… III. ①图灵（1912~1954） —人物研究 IV. ①K835.616.16

中国版本图书馆CIP数据核字（2012）第119110号

内容提要

图灵机是英国数学家阿兰·图灵提出的一种抽象计算模型，本书深入剖析了图灵描述图灵机和可计算性的论文《论可计算数及其在判定性问题上的应用》。书中在详解论文的同时, 附带了大量的历史背景资料、图灵的个人经历，以及图灵机对人们理解计算机、人类意识和 宇宙所产生的影响。

本书适合所有计算机科学专业的学生、程序员或其他技术人员，同时也适合欲了解图灵 生平以及他构建图灵机的思维过程的读者阅读。

图灵的秘密：他的生平、思想及论文解读

◆ 著 [美] Charles Petzold
译 杨卫东 朱 皓 等
责任编辑 傅志红
◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编 100061　电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京　　　印刷
◆ 开本: 700 × 1000　1/16
印张: 22.5
字数: 428千字 2012年11月第1版
印数: 1 -4 000册 2012年11月北京第1次印刷
著作权合同登记号 图字：01-2009-5735号
ISBN 978-7-115-28214-9

定价：69.00元
读者服务热线：（**010**）**51095186**转**604**　　印装质量热线：（**010**）**67129223**
反盗版热线：（**010**）**67171154**

版 权 声 明

图灵机（Türing Machine）：1936年，阿兰·图灵写了一篇关于计算机器的设计和局限性的论文，由此，它们被冠以了图灵的名字。这个变音ü其实是不必要的，这只是大家推测，这么难以理解的东西一定是德国人写的。

——摘自 *Faster Than Thought: A Symposium on Digital Computing Machines*（1953）

引 言

研究过计算机的历史、技术或理论的人，都会接触到“图灵机”这个概念。在1936年，为帮助解决数理逻辑中的一个问题，英国数学家阿兰·图灵（1912—1954）提出了图灵机。它是一种纯属虚构的计算机，连计算机假设也算不上。而由此得到的意外收获是，图灵创立了一个新的研究领域——计算理论（或可计算性），它主要研究数字计算机的功能和局限性。

尽管图灵机是一种并不太合理的计算机，但由于其自身极其简单而大放异彩。最基本的图灵机只能进行一些简单的操作。如果连这些操作都不能做，那么这台机器干脆什么都别做了。然而，只要将这些简单的操作组合起来，图灵机就能够进行现代数字计算机可以执行的任何计算。

拨开云雾见天日，通过考查计算机的原始基础，我们就能够更好地理解数字计算机的能力和局限性，这二者同样重要。尽管有人早就论证过计算机可以做什么，但在这种论证出现多年之前，图灵就证明了计算机永远都做不到的事。

图灵机仍然是被阐述和探讨的热门话题，你可以试试用喜爱的网络搜索引擎搜索“图灵机”。然而，我猜很少有人会阅读阿兰·图灵描述他这项创造的原始论文。或许，这与论文的标题“On Computable Numbers, with an Application to the Entscheidungsproblem”（“论可计算数及其在判定性问题上的应用”）有关。即使你会读最后那个单词（试试看，将重音放在第二个音节上，把这个音节发成类似“shy”的音，这就差不多了），并且知道它的意思（即判定性问题），你可能也会担心，图灵一定指望他的读者对繁冗的德国数学问题有基本的了解。快速浏览这篇论文（其中还用到了德国哥特式字体来表示机器状态）也无法让人消除这种担心。今天的读者还能手捧70年前伦敦数学学会集刊中的文章，并坚持看到有所收获，甚至十分满意吗？

这本书要讲的正是这篇论文。它包含了图灵原版36页的论文[1]“On Computable Numbers, with an Application to the Entscheidungsproblem” 和增补的3页修订[2]，并辅以背景材料和大量注解。阅读图灵的原版论文就是在探索他构建图灵机的思维过程，就像在

他充满想象、内容丰富的思想中进行一次奇特的旅行。图灵机不仅对计算产生了深远的影响，还深深影响了我们对数学局限性、人类思维方式，甚至宇宙本质的理解。（当然，图灵的论文中并没有出现“图灵机”这个术语，他称之为“计算机器”。不过，早在1937年[3]人们就开始使用“图灵机”这种说法，并且至今仍是标准术语。）

我在对图灵论文进行注释的过程中，发现用解释和阐述频繁打断他的叙述还是很有用的。我努力做到（但并没有完全做到）不打断他的某一整句话。大部分情况下，我会在讨论中保留图灵自己的术语和符号，不过有时，虽然图灵没有采用某个术语，如果我觉得这个术语在解释其工作时很有用，也会引入这些术语。

图灵论文的内容会像下面这样表示。

> We shall avoid confusion by speaking more often of computable sequences than of computable numbers.

> 为了避免混淆，我们会更多地提及可计算序列，而非可计算数。

我们（指出版商和我）努力保留图灵原始论文的字体和版式，[4]除非有一些奇怪的表示方法（比如冒号前加空格）在现代文字处理软件中总报错。原稿中所有的行间距也得以保留。图灵的论文中存在一些印刷错误、技术性错误和理论上的疏漏，尽管我没有在原文中加以修正，但会在评注中一一指出。图灵对他自己论文内容的引用，仍沿用原发表期刊中的页码，我没有修改这些引用，不过在评注中指出了被引用部分在本书中的页码。偶尔，你会在图灵的论文中发现一个括起来的数字，例如：

> When the letters are replaced by figures, as in § 5, we shall have a numerical
>
> [243]
>
> description of the complete configuration, which may be called its description number.

如果用数字代替这些字母，如在§5中，那么我们可以得到这个完全格局的数字表示，也可以称作它的描述数。

这是原论文的分页处以及标注的页码。我这本书的脚注采用的是圆圈编号，而图灵论文的脚注使用符号标注，并写在阴影部分。

如果只保留本书阴影部分的英文内容，再组合起来，得到的就是完整的图灵论文，而我这个劳而无功的作者只能欲哭无泪了。更有趣的阅读方式是，先读本书，再读没有被我打断的图灵论文。

图灵的论文分散在本书的第4~15章，其修订内容在第16章。他的论文分为11个部分和一个附录，对应到本书的页码是：

图灵写这篇论文的最初动机是想解决德国数学家大卫·希尔伯特（1862—1943）构想的一个问题。希尔伯特想寻找一种通用的方法来判定数理逻辑中的任意命题是否可证。寻找这种“通用的方法”被称为判定性问题。尽管判定性问题确实是图灵写这篇论文的动机，但是这篇长篇大论本身讲的却是可计算数。在图灵的定义中，可计算数就是可以使用机器计算的数。论文前面60%的内容都是图灵对可计算数的探索，就算完全不了解希尔伯特在数理逻辑或判定性问题方面的研究，也能够阅读并理解这些内容。

了解可计算数与“实数”的区别对于理解图灵的观点很重要。因此，本书利用前几章介绍了数字分类的背景知识，数字包括整数、有理数、无理数、代数数和超越数，它们都可归为实数。我尽可能不涉及比高中

数学更复杂的知识。我知道，有些读者离开快乐的高中生活已经几十年了，我要努力唤醒这些记忆。如果由于我本着这种教育热情而做出一些冒犯读者的解释，我表示歉意。

尽管我觉得本书的读者大多会是计算机科学专业的学生、程序员或其他技术人员，但是我还是尽量让非程序员的读者也愿意读，因此我定义了一些便于理解的术语。图灵的论文被誉为“20世纪的一座知识地标”[5]，我希望本书可以让更多的读者领略到这篇论文的风采。

为了满足不同读者的需要，本书分成了四个部分。

第一部分“基础”介绍阅读图灵论文所必须掌握的一些历史和数学背景知识。

第二部分“可计算数”包含了图灵论文的大部分内容，也是关心图灵机和可计算性相关问题的读者最感兴趣的部分。

第三部分“判定性问题”先简要介绍了数理逻辑的背景知识，然后讨论图灵论文的剩余部分。

第四部分“题外话”讨论了图灵机为何成为人们理解计算机、人类意识和宇宙本身的必要工具。

第三部分的数学内容肯定是比前几章的难，并且讲得比较快。对图灵论文在数理逻辑方面的影响不感兴趣的读者甚至可以跳过第三部分，直接阅读第四部分。

本书涉及数学中几个大的研究领域，包括可计算性和数理逻辑。我仅仅把与理解图灵论文最相关的那些主题和概念挑出来加以解释，省去了很多细节，因此本书从深度和严格性上都无法取代那些可计算性和逻辑方面的专业书籍。想深入研究这些领域的读者可以查阅参考文献。

阿兰·图灵一生发表过近30篇论文和文章[6]，却从未写过书。其中的两篇论文造就了他流芳百世的声望。“On Computable Numbers”（“论可计算数”）当然是第一篇。第二篇名为“Computing Machinery and Intelligence”（“计算机器和智能”，发表于1950年），这一篇的技术性不是很强，图灵在文中首次提出了一种判断人工智能的标准，在今天被称为“图灵测试”。总的来说，一台机器如果可以骗得我们相信它是一个人，那么就可以说它是智能的。

图灵机和图灵测试是阿兰·图灵声名不朽的两大基石。初看上去，它们像是两个完全不同的概念，但事实并非如此。图灵机是以一种非常机械的方式展现人类如何进行数学运算的，图灵测试则是对计算机能力的人为评估。在整个数学研究期间，图灵都在探索人类思维和计算机器之间的关系，他所采用的研究方法至今仍很吸引人。

很多关于可计算性的教科书只讨论图灵的研究而不涉及图灵这个人，它们可没有劳神讲述有关个人传记的细节。不过，本书不会这么

做。图灵在二战期间所做的密码分析方面的秘密工作，他参与的影响力巨大的计算机工程，他对于人工智能的思索，他的性取向，他由于“严重猥亵”罪而被逮捕和起诉的经历，以及他在41岁时自杀身亡，所有这些事情都需要关注。

得益于英国数学家安德鲁·霍奇斯（1949—）撰写的精彩传记 *Alan Turing: The Enigma*（《艾伦·图灵传：如谜的解谜者》，Simon & Schuster，1983年出版），我没费多大力气就总结出了图灵一生中的重要事件。霍奇斯对图灵感兴趣的部分原因，在于他参与了20世纪70年代的同性恋解放运动。霍奇斯的传记还给休·怀特摩尔的剧本*Breaking the Code*（《破解密码》，1986）带来了灵感，在舞台上和在1996年改编的电视片中，阿兰·图灵的角色都是由德里克·雅克比扮演的。

如同早期的英国数学家、计算机先驱查尔斯·巴贝奇（1791—1871）和艾达·拉夫拉斯（1815—1852），图灵也成为计算机时代的一个标志。美国计算机协会每年都会为在计算机行业做出杰出贡献的人颁发图灵奖，奖金为10万美元。现在还有一些用来组装图灵机的工具，比如“图灵编程语言”（从Pascal衍生而来）和“图灵的世界”软件。

图灵的名字几乎成为计算机编程的通用代名词。杜特尼把他的“计算机科学探索”一书命名为*The Turing Omnibus*（《图灵选集》，计算机科学出版社，1989）。戴维德·波尔特把他编写的一本关于“计算机时代的西方文化”的书命名为*Turing’s Man*（《图灵时代的人类》，北卡罗来纳州大学出版社，1984）。布莱恩·罗特曼对传统数学极限概念的评论文章*Ad Infinitum*（斯坦福大学出版社，1993）被幽默地加上了副标题*The Ghost in Turing’s Machine*（《图灵机里的幽灵》）。

数学和计算机科学领域以外的学者也对阿兰·图灵感兴趣。研究文集*Novel Gazing: Queer Readings in Fiction*（《凝神注视：论小说的另类解读》）中最有特色的一篇文章就是由泰勒·科坦撰写的 *The “Sinister Fruitiness” of Machines: Neuromancer, Internet Sexuality, and the Turing Test*（《智能机器带来的“阴暗苦果”：神经漫游者、网络性爱和图灵测试》）。科坦博士所说的*Neuromancer*指的是威廉·吉布森著名的“赛博朋克”小说*Neuromancer*（《神经漫游者》）。在这部科幻小说里，有一个叫做图灵警察局的组织，他们负责确保人工智能体不会试图增强它们自身的智能。

图灵还出现在很多小说的书名中。马文·明斯基（麻省理工学院人工智能方向著名的研究者）与科幻小说家哈里·哈里森合写了*The Turing Option*（《图灵选择》，华纳图书公司，1992）。伯克利计算机科学教授克里斯托斯·帕帕迪米特里欧参与创作了*Turing*（《图灵》，一部关于计算的小说，麻省理工学院出版社，2003）。

玻利维亚小说家埃德蒙多·苏丹写了一本名为*Turing’s Delirium*（《图灵的狂热》，英文版由丽莎·卡特翻译，霍顿·米夫林出版公司，2006）的小说，在其中，一个外号叫图灵的密码专家发现了用他的技能为腐败政府服务带来的危险。在珍娜·列文的小说*A Madman Dreams of Turing Machines*（《图灵机狂人梦》，Knopf出版社，2006）中，阿兰·图灵和库尔特·哥德尔的生活被虚构在了一起，他们穿越时空，产生了奇特的交织。

阿兰·图灵这个角色还出现在其他很多小说中，如尼尔·斯蒂芬森的*Cryptonomicon*（《编码宝典》，Avon，1999），罗伯特·哈里斯的*Enigma*（《密码迷情》，Hutchinson，1995），约翰·卡斯蒂的*The Cambridge Quintet: A Work of Scientific Speculation*（《剑桥五重奏：一部科学思考的著作》，Perseus图书公司，1998），以及道格拉斯·侯世达的*Gödel, Escher, Bach*[7]（Basic图书公司，1979）。阿兰·图灵甚至为*The Turing Test*（《图灵测试》，BBC，2000）的一部分做了解说，这本书是保罗·伦纳德写的Doctor Who系列小说中的一本。

人们以各种方式来表达对阿兰·图灵的尊敬当然是好事，不过这样一来，图灵的实际研究可能会被遗忘。我希望，就算那些正式研究过计算理论，并认为自己完全了解图灵机的人，也能在面对这个真正由大师自己构建的图灵机时发现不少令人惊奇的事物。

* * *

我在1999年就开始构思这本书，当时只写了一点，然后在接下来的五年里时不时又写上一些。2004～2005年基本完成了前11章。后面7章是在2007～2008年完成的，在此期间的写作几乎未中断，唯一的中断就是与我一生中最好的朋友，也是我的至爱迪尔德丽·辛诺特结婚（终于结婚啦）！

非常感谢伦敦数学协会许可完整地再版阿兰·图灵的论文“On Computable Numbers, with an Application to the Entscheidungsproblem”。

沃尔特·威廉姆斯和拉里·史密斯审阅了本书的初稿，发现了一些错误，并且提出了一些很有益的改进建议。

非常感谢Wiley出版公司的同仁，正是他们的工作将我所钟爱的想法真正出版成书。克里斯· 韦伯负责督促这本书的出版，策划编辑克里斯多夫· 里韦拉和制作编辑安吉拉·史密斯克服了很多版式和印刷方面的困难，技术编辑彼得·伯凡蒂帮助我认真完成了技术相关的内容。Wiley出版公司的很多幕后工作人员也都努力把这本书做得至臻至善。所有未被发现而遗留在书中的缺陷、瑕疵或隐藏的错误，都只能归咎于作者。

每位作者都是站在前人肩上的。选出的参考书目只列出了我所参考的众多书籍中的一小部分。我还要感谢纽约公共图书馆，特别是科学、

工业和商业图书馆的工作人员。为参考原始论文，我多次使用JSTOR，同时我发现维基百科、谷歌书籍搜索和Wolfram MathWorld也都很有用。

*　*　*

登录网站www.TheAnnotatedTuring.com可以找到与本书相关的信息和资源。

查里斯·佩措尔德
纽约州纽约市和罗斯科
2008年5月

---

[1] 阿兰·图灵，“On Computable Numbers, with an Application to the Entscheidungsproblem”，*Proceedings of the London Mathematical Society*，2nd series，Vol.42（1936），pp.230-265。

[2] 阿兰·图灵，“On Computable Numbers, with an Application to the Entscheidungsproblem A Correction”，*Proceedings of the London Mathematical Society*，2nd series，Vol.43，(1937)，pp.544-546。

[3] 阿隆佐·邱奇对“On Computable Numbers, with an Application to the Entscheidungsproblem”，一文的评论，*The Journal of Symbolic Logic*，Vol. 2，No. 1，1937年3月，42-43。

[4] 中文版依然保留图灵原始论文的印刷体和排版，其下是中译文，译文不再保留页码和版式。——编者注

[5] 这一说法是John P. Burgess在George S. Boolos、John P. Burgess和Richard C. Jeffrey所著的 *Computability and Logic*, fourth edition（Cambridge University Press，2002）一书的前言中提到的。

[6] 这些和其他文档可以从*The Collected Works of A.M. Turing*（Amsterdam: Elsevier，1992，2001）的4卷中获得。其中大部分重要资料由B. Jack Copeland收集到 *The Essential Turing*（Oxford University Press，2004）和 *Alan Turing’s Automatic Computing Engine*（Oxford University Press，2005）中。前者包含了与图灵机相关的文章和论文，后者是关于20世纪40年代中后期ACE计算机工程的。

[7] 中文版《哥德尔、埃舍尔、巴赫——集异璧之大成》由商务印书馆1996年8月出版。

——译者注

目 录

参考文献

第一部分
基 础

第1章
这个墓穴埋葬着丢番图

在很多个世纪以前的古亚历山大，一位老人埋葬了自己的儿子。这位心碎的老人为了转移自己的悲伤，开始整理大量的代数问题，并将这些问题及其解法汇编成书，取名《算术》（*Arithmetica*）。这些就是人们对亚历山大的丢番图几乎所有的了解，而这些了解绝大多数来自其好友在他去世后不久所写的一个谜题：[1]

> 行人啊，请稍驻足，这里埋葬着丢番图。上帝赋予他一生的六分之一，享受童年的幸福；再过十二分之一，两颊长胡；又过了七分之一，燃起结婚的蜡烛。爱子的降生盼了五年之久，可怜那迟来的儿郎啊，只活到父亲岁数的一半，便进入冰冷的坟墓。悲伤只有通过数学来消除，四年后，他自己也走完了人生旅途。[2]

这篇墓志铭对丢番图儿子的死亡说得不是很清楚。其中提到，他只活到了“父亲岁数的一半”，但这是指儿子死时父亲年龄的一半，还是指他父亲寿命的一半？不论怎样理解，都可以解答。但如果是后一种理解“只活到他父亲寿命的一半”，我们得出的岁数会是一个漂亮而又简洁的整数。

我们假设丢番图的寿命为x。丢番图生命中每个时期的年数要么是他寿命的几分之几（例如，x除以6是他的童年时光），要么是一个整数（例如，从他结婚到儿子出生有5年的时光）。丢番图生命中所有时期的年份之和为x，所以这个谜题可以用下面这个简单的代数式来表示：

$$\frac{x}{6}+\frac{x}{12}+\frac{x}{7}+5+\frac{x}{2}+4=x$$

所有分母的最小公倍数是84，将等号两边同时乘以84得到：

$$14x+7x+12x+420+42x+336=84x$$

分别整理带有x的项和常数项，得到：

$$84x-14x-7x-12x-42x=420+336$$

即：

$$9x=756$$

方程的解是：

$$x=84$$

所以，丢番图的童年时光是14年，7年后他长大成人。又过了12年，在33岁的时候，他结了婚，5年后有了儿子。儿子死于42岁，丢番图当时80岁，4年后丢番图去世。

事实上，有一个更快捷的方法来解这个谜题：如果深入探索出题人的内心想法，你就会发现他并不想用分数来增加麻烦。丢番图寿命的“十二分之一”和“七分之一”必然是整数，所以他的寿命年数一定可以被7和12整除（自然也会被2和6整除）。只需将12乘以7就能得到84。这个看起来也像是合适的高龄岁数，所以它极有可能是对的。

丢番图去世时也许是84岁，但是对于历史来说，更重要的问题是找到具体时间。人们曾经猜测，丢番图的时代是在公元前150年到公元280年之间[3]，那是一个令人向往的时期。这样的话，丢番图就活在欧几里得（活跃在约公元前295年[4]）和埃拉托色尼（约公元前276—前195年）等早期亚历山大数学家们之后，这也说明他与亚历山大的海伦（活跃在公元62年）处于同一时期。海伦的著作涉及了力学、气体力学以及自动控制，他似乎还发明了一种原始蒸汽机。丢番图也许还认识那位凭著作《天文学大成》而被世人铭记的亚历山大天文学家托勒密（约公元100—170）。那本书包含了世界上第一个三角函数表，并且建立了直到十六七世纪哥白尼革命时才被推翻的描述天体运动的数学。

不幸的是，丢番图也许从未见过这些亚历山大的数学家和科学家们。过去一百多年来，古典学者们之间的共识是，丢番图大约活跃在公元250年，他现存的主要著作《算术》很可能也追溯到那个时期。这样的话，丢番图的出生时间大概是在托勒密去世时间的前后。曾经编辑了权威的希腊版《算术》（1893~1895年出版）的保罗·塔纳里注意到，这本书写着献给“尊敬的狄奥尼修”。虽然这是一个常用名，但塔纳里猜测，这个狄奥尼修就是那个曾在公元232~247年担任亚历山大传道学校校长，以及之后在公元248~265年担任亚历山大主教的狄奥尼修。因此，丢番图可能是个基督徒。[5]如果是这样，下面这一事实就有点讽刺意味了：对《算术》的一个早期但遗失了的评注是由塞翁的女儿希帕蒂亚（约公元370—415）所写的，她是亚历山大最后一位伟大的数学家，后来被一帮反对她“异教徒”哲学思想的基督教暴徒杀害。

古希腊数学家在几何学和天文学领域一直是最强的。丢番图在种族上是希腊人，但与众不同的是，他用“数字的科学”，即我们所知的代数，来缓解儿子去世的悲痛。他似乎是代数上很多创新的源头，包括他在问题中使用的符号和缩写，这标志着数学问题从文字描述到现代代数

表示法的转变。

《算术》的6本书（原来是13本）中罗列的问题一道比一道难，大部分都难于求解丢番图年龄的问题。丢番图的问题常常含有多个未知量。他的一些问题是不定的，也就是说这些问题通常有多个解。《算术》中只有一个问题不是抽象的，也就是说其他问题都是绝对数字化、不指代现实事物的。

丢番图提及的另一个抽象元素是幂。那个时候，数学家们已经熟悉了平方和立方。平方用来计算一个平面图形的面积，立方用来计算一个实体的体积。但是丢番图将高次方引入了他的问题：4次方（他称为“平方-平方”）、5次方（他称为“平方-立方”）和6次方（他称为“立方-立方”）。丢番图知道，这些幂与现实没有关联性，并且他也不在乎这种数学的实用性。这是纯粹的娱乐性数学，仅仅用来强化思维，没有别的目的。

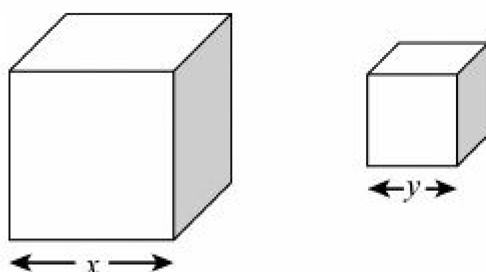
这里列举第4本书中的第一个问题。[6]丢番图先是概括地阐述了：

> 将一个已知数拆分成为两个立方体的体积，并且这两个立方体的边之和等于另一个已知数。

接着给出了例子：

> 已知数为370，边长之和是10。

将这个问题用图表示后可见，他需要处理两个不同边长的立方体。现代代数学家可以将这两个立方体的边标记为x和y：



这两条边加起来为10。这两个立方体的体积之和（x^3和y^3）是370。我们现在写下两个等式：

$$x+y=10$$
$$x^3+y^3=370$$

由第一个等式得出，y等于$(10-x)$，将其代入第二个等式：

$$x^3+(10-x)^3=370$$

展开$(10-x)^3$，我们希望立方项最终可以消失：

$$x^3+(1000+30x^2-300x-x)^3=370$$

很幸运，立方项消失了，经过整理后可以得到：

$$30x^2-300x+630=0$$

等式左边的3个数有一个公因数，所以可以同时除以30：

$$x^2-10x+21=0$$

现在，这个问题基本解决了。你有两个选择。如果记得二次方程的求根公式[7]就可以直接使用它；或者，如果你曾经练习过求解类似的方程，就可以一直盯着它思索，直到它自己神奇地分解成

$$(x-7)(x-3)=0$$

因此两个边的长度分别为7和3。的确，这两个边加起来等于10，它们的立方（343和27）和等于370。

丢番图并不像你我这样解决这个问题，他确实不会。尽管丢番图的问题经常涉及多个未知数，但是他的记号只允许他表达一个未知数。他用了一个巧妙的方法弥补了这一点。他没有将两个立方体的边长标记为x和y，而是标记为$(5+x)$和$(5-x)$。这两个边长可以用一个未知数x表示，并且加起来确实等于10。接下来，他就可以将这两条边进行立方运算，相加后等于370：

$$(5+x)^3+(5-x)^3=370$$

这个式子看起来比我们的糟，但是如果展开这些立方，一些项便会迅速消去，只留下：

$$30x^2+250=370$$

合并同类项，方程两边再同除以30，进一步化简为：

$$x^2=4$$

即$x=2$。因为两条边是$(5+x)$和$(5-x)$，所以这两条边是7和3。

丢番图用来解决这个问题的方法比现在学生用的方法轻松，他神奇并正确地将两个边长用一个未知数表示。这个方法会适用于下一个问题吗？也许可以，也许不可以。建立解决代数方程的通用方法确实不是丢番图所要考虑的。正如一位数学家论述的：“每一个问题都需要一个十分具体的方法，这个方法通常连最类似的问题都不适用。这使得现代数学家即使在研究了100道丢番图问题的解答后，还是很难找到解决第101道题的方法。”[8]

当然，丢番图在展示这个立方之和为370、边长之和为10的问题时，显然并不是随意选取某些数字，他知道这些假设条件将会导出一个整数解。实际上，丢番图方程就是指只允许整数解的代数方程。丢番图方程可以有很多未知量，这些未知量可以带有整数幂，但是它的解（如果有）总是整数。尽管丢番图经常使用减法来命题，但是他的解从不涉及负数。“对于一个没有用任何正整数相减就得到的负整数本身，丢番

图显然没有任何概念。”[9]任何一道问题也不会包含有0的解，古希腊人不将0考虑在内。

现代读者们，特别是那些已经默认了丢番图问题只有整数解的人，在遇到丢番图问题中的有理数时也许会有点吃惊。有理数之所以这样命名，不是因为它们在某种程度上符合逻辑，而是因为它们可以表示为两个整数的比。例如：

$$\frac{3}{5}$$

就是一个有理数。

在《算术》中，有理数只出现在涉及现实物体的问题中，特别是那些一直被大家津津乐道的问题：饮料和德拉克马（古希腊货币）。虽然从这个问题的描述里看不出来，但是有理数在这个解中是必需的：

> 一个人买了若干份酒，有些单价是8德拉克马，有些是5德拉克马。他为这些酒支付的德拉克马是个平方数，如果这个数再加上60，结果还是一个平方数，该平方数的根是这些酒的份数。求两类酒他各买了多少。[10]

这里的“平方数”是指一个数与它自身的积。例如，25是一个平方数，因为它等于5乘以5。

在进行了一整页的计算后，[11]它揭示了单价5德拉马克的数量是一个有理数：

$$\frac{79}{12}$$

单价8德拉马克的数量也是一个有理数：

$$\frac{59}{12}$$

我们检验一下这个结果。（检验这个结果要比推导它容易得多。）如果你用5德拉马克乘以79/12，然后加上8德拉马克乘以59/12的积，就会发现这个人总共支付了$72\frac{1}{4}$德拉马克。丢番图说这个人支付了“平方数的钱”。支付的钱数必须是某个数的平方。令人好奇的是，丢番图认为$72\frac{1}{4}$是个平方数，因为它可以表示为：

$$\frac{289}{4}$$

分母和分子都是平方数：分别是17和2的平方。因此，$72\frac{1}{4}$是$\frac{17}{2}$（即$8\frac{1}{2}$）的平方。丢番图进一步说：“如果这个数再加上60，结果还是一个平方数，该平方数的根是整个酒的数量。”这里的“整个”不是指整数。丢番图（或者说是《算术》英文版的译者托马斯·哈斯爵士）的意思是指度量的总份数。60加$72\frac{1}{4}$是$132\frac{1}{4}$，也就是有理数：

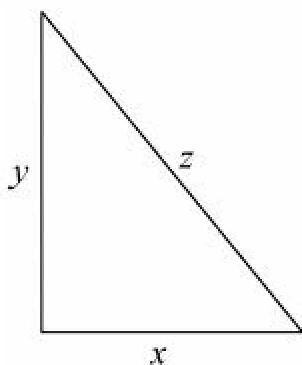
$$\frac{529}{4}$$

丢番图再一次认为这个数是平方数，因为它的分子和分母都是平方数：分别是23和2的平方。因此，总的度量数是23/2（即$11\frac{1}{2}$），这同样可以通过将79/12和59/12相加得到。

《算术》中最著名的问题也许要算第2本书的第8个问题：将给出的平方数分解为两个平方数的和，也就是说，求x、y、z，使它们满足：

$$x^2+y^2=z^2$$

这个问题的几何解释是毕达哥拉斯定理所描述的直角三角形三条边之间的关系。



这个问题有许多整数解，例如x、y、z分别等于3、4、5（两个平方数9和16的和等于25）。这个简单的结果显然不是丢番图所希望的。他设定了一个“给出的平方数”（也就是z^2）等于16，于是其他两边分别等于144/25和256/25。对于丢番图来说，这些数当然都是平方数，其中第一个数是12/5的平方，第二个数是16/5的平方，并且它们的和是4的平方：

$$\left(\frac{12}{5}\right)^2+\left(\frac{16}{5}\right)^2=4^2$$

丢番图允许有理数解并不重要，因为这个解等价于一个整数解。简单地将等式两边同乘以5^2（即25），即可得到：

$$12^2+16^2=20^2$$

即144加256等于400。事实上，这是同一组解，它们的不同仅在于度量边的方式不同。丢番图的问题阐述中，斜边是4。这可能是4英尺。现在用一个单位长度不同的尺子去测量，比如单位长度等于五分之一英尺。用这个尺子测量，这条斜边就等于20，其他两条边分别为12和16。

整数是在人们开始计数之时出现的，有理数也许是在人们开始测量时出现的。如果一根胡萝卜的长度等于3根手指的宽度，另一根胡萝卜的长度等于4根手指的宽度，这时第一根胡萝卜的长度就是第二根的$\frac{3}{4}$。

有理数有时也称为可通约数字，因为长度被表示成有理数的两个物体总可以重新度量为整数长度，你只需要将新的度量单位变得足够地小。

丢番图的《算术》是用希腊语写的，至少有部分文稿被翻译成了阿拉伯文。当它开始在欧洲数学界产生影响的时候，在1575年首次被翻译成拉丁语，之后在1621年有了更好的版本。费马（1601—1665）曾拥有一本1621年的拉丁语版《算术》，并在其空白处写满了笔记。1670年，费马的儿子公布了这些笔记以及拉丁文版的《算术》。在这道问题旁有这样一段笔记，费马写道：

> 另一方面，将一个立方数分解为2个立方数，或者将一个4次方数分解为两个4次方数，亦或将除平方之外的任何乘方分解为两个有同幂的乘方，这些都是不可能的。对此，我已经发现了一个非常漂亮的证明，但是这儿的空白之处不够写下它。[12]

费马宣称，例如：

$$x^3+y^3=z^3$$

是没有整数解的，并且幂为4、5、6及之后的类似方程都没有解。这并不明显。等式：

$$x^3+y^3+1=z^3$$

非常接近于

$$x^3+y^3=z^3$$

而且它有许多整数解，例如x、y、z分别等于6、8、9。等式

$$x^3+y^3-1=z^3$$

同样相似，也有许多整数解，例如9、10、12。为什么这两个相似的等式有解，但是

$$x^3+y^3=z^3$$

没解呢？

丢番图在《算术》中介绍的问题都有解，但是许多丢番图方程，例如费马描述的方程，看起来并没有解。对于数学家来说，确定一个丢番图方程是否有整数解比求解特定的丢番图方程更加有趣。

费马没有写出的证明就是大家熟知的费马最后定理（有时也称费马大定理）。多年来，人们普遍相信，不管费马当时想到了怎样的证明，这个证明也许都是错的。英国数学家安德鲁·怀尔斯（1953—）从10岁开始就对这个问题产生了兴趣，到了1995年，费马最后定理才最终被他证明。（人们很早就证明了，对于一些特殊情况，例如指数为3时，方程是无解的。）

很显然，证明某些丢番图方程没有解要比找到一个解（如果有）更具挑战性。如果你知道某个特定的丢番图方程存在解，可以简单地验证所有的可能性。由于允许的解只能是整数，因而你可以首先尝试1，然后是2、3及之后的数。如果你不想做这些繁重的工作，可以写一个计算机程序测试所有的可能性，程序迟早会帮你找到答案的。

但是，如果并不知道是否存在解，那么这个用计算机蛮力解决的方案就不合适了。你可以不断尝试，但怎样知道何时该放弃呢？你怎么知道下一步将要测试的一组数字不是所要搜寻的那组数字呢？

麻烦来自这些可恶的数字：它们有无穷多个。

---

[1] 托马斯·希恩，*Diophantus of Alexandria: A Study in the History of Greek Algebra*，second edition（Cambridge University Press，1910，Dover Publications，1964），3。

[2] *Greek Mathematical Works II: Aristarchus to Pappus of Alexandria*（Loeb Classical Library No. 362），由Ivor Thomas翻译（Harvard University Press，1941），512–3。

[3] 这些日期来自Simon Hornblower and Antony Sprawforth，eds.，*Oxford Classical Dictionary*，revised third edition（Oxford University Press，2003），483。

[4] 这些亚历山大数学家们的生活年代来自Charles Coulston Gillispie，ed.，*Dictionary of Scientific Biography*（Scribners，1970）。

[5] 希恩，*Diophantus of Alexandria*，2，note 2。希恩本人好像也对此持

怀疑态度。

[6] 希恩，*Diophantus of Alexandria*，168。

[7] 对于$ax^2 + bx + c = 0$，解为$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$。

[8] Hermann Hankel (1874) 引用希恩的 *Diophantus of Alexandria*，54–55。其他数学家就丢番图的方法找到了更清晰的方式。参见Isabella Grigoryevna Bashmakova，*Diophantus and Diophantine Equations*（Mathematical Association of America，1997），ch. 4。

[9] 希恩，*Diophantus of Alexandria*，52–53。

[10] 希恩，*Diophantus of Alexandria*，224。

[11] 希恩，*Diophantus of Alexandria*，225。

[12] 希恩，*Diophantus of Alexandria*，144，note 3。

第2章
无理数和超越数

从1，2，3…开始数起，只要我们愿意就可以一直数下去。这些数就是我们熟知的基数、整数，或者说自然数。它们看起来确实相当自然，因为宇宙中有很多我们可以计数的物体。自然数也许是早期人类想象出的第一个数学对象。一些动物似乎也有数的概念，只要这些数不是太大。

几百年来，零一直不算作自然数，甚至至今也没有共识。（在教科书的数论部分，作者通常会在第一页标明是否将零归入自然数。）负整数位于零的另一侧，正整数、负整数及零共同构成了整数集合。整数在正负两个方向趋向无穷大：

… -3 -2 -1 0 1 2 3 …

我们将从1开始的所有正的整数称为正整数。对于那些从零开始的正整数集合（即0，1，2，3，…），可以称为非负整数，既明确，也不会太冗长。

有理数是可以表示为两个整数之比的数，但是分母不能为零。例如，

$$\frac{3}{5}$$

是一个有理数，它也可以写成小数形式：

0.6

有理数包含所有整数，因为任何整数（例如47）都可以写成分母为1的分数形式：

$$\frac{47}{1}$$

任何有限小数也是有理数。例如，

-23.45678

可以写成比的形式：

$$\frac{-2345678}{100000}$$

有些有理数，例如：

$$\frac{1}{3}$$

的小数形式会写成无限循环小数：

$$0.3333333333\ldots$$

因为它能写成比的形式，所以仍然是个有理数。实际上，任何无限循环小数都是有理数，下面这个数，

$$0.234562345623456\ldots$$

如果23456会重复出现，那么它就是个有理数。为了证明这个数是有理数，我们用x表示这个数

$$x=0.234562345623456\ldots$$

然后等式两边同时乘以100000：

$$100000x=23456.23456234562346\ldots$$

我们知道，等式两边同时减去相同数值，等式仍然成立。也就是说，在第二个等式中，可以让等式两边的数同时减去第一个等式的数：令10000x和23456.23456…分别减去x和0.23456…，这样分数部分就消失了：

$$99999x=23456$$

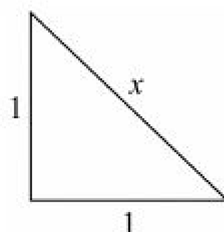
所以：

$$x=\frac{23456}{99999}$$

这是一个整数比，所以它是有理数。

大致看来，有理数集似乎是完备的。如果把两个有理数相加，结果还是有理数；同样，将有理数相减、相乘或相除，结果仍然是有理数。

也许有人会想（就像以前人们那样），所有的数都是有理数，但是如果考虑这个直角三角形的斜边：



根据勾股定理，

$$x^2=1^2+1^2$$

即：

$$x^2=2$$

也即：

$$x=\sqrt{2}$$

是否存在某个整数与整数的比值，当它乘以自身时等于2？当然，人们可以找到许多非常接近的有理数。下面就是个例子：

$$\frac{53492}{37825}$$

这个有理数很接近，只差一点点了。当它乘以自身时得到1.999 95。如果我们继续寻找，也许可以找到一个完美的答案。

但也许我们这样做只是在浪费时间？

证明一些东西不存在是很困难的，但是数学家们发明了一种在类似情况下巧妙解决问题的证明方法。这个方法叫做间接证法，又称归谬法、背理法。先提出一个假设，然后根据这个假设进行符合逻辑的推理，直到推出一个矛盾的结论。这个矛盾的结论说明我们最初的假设是错误的。

归谬法看起来拐弯抹角，但是它在现实生活中的应用也许比我们想象的更普遍，“不在场证明”就是一种归谬法。如果被告人被怀疑在犯罪现场，而案件发生时他在自己母亲的家里，那么就意味着他在同一时间出现在了两个地方，这是荒谬的。

我们假设2的平方根是有理数。因为它是有理数，所以存在整数a和b，使得：

$$\frac{a}{b}=\sqrt{2}$$

a和b是否都是偶数？如果是，同时除以2并且用得到的数来代替a和b。如果得到的数仍然是偶数，再除以2，一直持续做，直到a和b至少有一个是奇数。

等式两边同时平方：

$$\frac{a^2}{b^2}=2$$

即：

$$a^2=2b^2$$

注意，a的平方是b的平方的2倍，这就说明a的平方是个偶数，而要想a的平方为偶数，a必须为偶数。先前我们推导出a和b不可能都是偶数，所以我们知道b是奇数。

如果a是偶数，它应该等于某个数的2倍，我们称这个数为c：

$$(2c)^2=2b^2$$

即：

$$4c^2=2b^2$$

也即：

$$2c^2=b^2$$

这说明b的平方是偶数，也就是说b是偶数，这与我们先前的假设“a和b不能都为偶数”相悖。

因此，原来的假设“2的平方根是有理数”是错误的。毫无疑义，2的平方根是无理数。当它以小数形式呈现时，这些数字将无序地排列下去

1.4142135623730950488016887242097…

这个数只能在有无限的纸张、无数的笔和无限的时间下才能准确地表达。我们只可能把它写成近似值，并用一个省略号来承认我们的失败。要想用有限的方式表达这个数，最贴切的方法是提供计算这个数的算法。（这正是我要在第6章详细阐述的。）

我们用“有理”和“无理”来形容一个数，仿佛数字真的会疯掉一样。其实，这是有历史原因的。有时无理数也称为“不尽根”（surds），这个词和“荒谬”（absurd）有关。古希腊人对无理数并不陌生，但不怎么喜欢它们。据说（无可靠历史依据），毕达哥拉斯的学生希帕索斯在公元前6世纪发现2的平方根是无理数。故事里还说，此发现引起了轩然大波，毕达哥拉斯及其追随者试图掩盖这一发现，甚至把希帕索斯扔进了地中海。他们相当肯定，无理数不存在。丢番图拒绝承认无理数是其问题的解，延续了无理数不合他口味的这一传统。

在有了小数点（古希腊人没有）后，我们就能轻易创造一个数，它显然是一个无理数，只要写下一些无重复片段的无序数字就行了。这样一个小数，它的小数部分出奇地怪异，但显然不会重复：

.00101101110111101111101111111…

在小数点之后，有两个0和一个1，然后一个0和两个1，再后一个0和三个1，等等。这不是一个有理数！它不能表示成两个整数的比，因此它是无理数。

2的平方根是方程：

$$x^2-2=0$$

的解。这个等式和先前展示的一样，只是我们将2移到了等号的另一边。17的立方根（同样也是个无理数）是方程：

$$x^3-17=0$$

的解。上面两个方程都叫做代数方程。下面是另一个代数方程：

$$-12x^5+27x^4-2x^2+8x-4=0$$

代数方程有一个变元，通常表示为x。（代数方程和丢番图方程不同，丢番图方程可以有多个变元。）代数方程有相加为零的多个项，上述最后一个例子里有五项。每一项包含一个变元的幂，幂为整数或零。（因为任何数的零次方都是1，第五项可以表示为-4乘以x的零次方。）任何带幂的变元都乘以一个整数系数，在这个例子中，系数依次是-12、27、-2、8和-4。这些系数可以是零，就像这个例子里"丢失"的x的立方项。

代数方程在现实问题中频繁出现，所以它们很受重视。代数方程的一般形式是：

$$a_N x^N + a_{N-1} x^{N-1} + \ldots + a_2 x^2 + a_1 x + a_0 = 0$$

其中，N是正整数，a_i是整数。它可以更简明地写成：

$$\sum_{i=0}^{N} a_i x^i = 0$$

在我们先前的例子：

$$-12x^5 + 27x^4 - 2x^2 + 8x - 4 = 0$$

中，N（最高的指数，也叫多项式的次数）是5，a_5是-12，a_4是27，a_3是0，等等。

代数方程的解（也叫方程的根）称为代数数。一个N次多项式最多可以有N个不同的解。在第1章，代数方程

$$x^2 - 10x + 21 = 0$$

有3和7两个解。

2的平方根是代数方程：

$$x^2 - 2 = 0$$

的一个解，2的负平方根是另一个解。

代数数的范畴还包括所有整数和所有有理数。例如，整数5是代数方程：

$$x - 5 = 0$$

的解，而3/7是代数方程：

$$7x - 3 = 0$$

的解。有些代数方程的解只有负数的平方根：

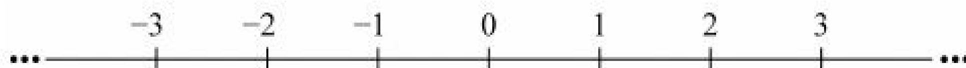
$$x^2 + 5 = 0$$

这个方程看起来是无解的，因为任何数乘以它本身都是正的量，加上5不可能得0。负数的平方根称作虚数。（为了方便，-1的平方根记作字母 i。）尽管名字如此，但虚数是一类非常有用的数，它在现实生活中有着广泛的应用。不过，图灵论文和本书并不涉及虚数。

在18世纪的某个时间，数学家们开始使用实数这一名称，以便同虚

数区分开。根据定义，实数包括了除负数平方根以外的一切数。

实数也称为连续统，因为实数可以看成一条连续直线上全体点的集合：



这条线上标记了一些整数，但是单靠这些整数点显然无法形成一条连续的线。

同样，有理数全体也不是连续的。显然，有理数在实数轴上看上去是非常稠密的。对于任意两个有理数，例如a和b，你都可以在它们之间插入另一个有理数，如a和b的平均数：

$$\frac{a+b}{2}$$

但是，在有理数之间仍存在无理数占据的间隙。例如，其中的一个间隙对应了2的平方根。

现在，我们从两个角度对数进行分类。我们已经将代数方程的解定义为了一类，称作代数数，这一类包括整数、有理数和许多如平方根和立方根的无理数。我们还定义了一类数，称作实数，它是除负数平方根外的其他数。现在的问题是：

所有的实数都是代数数吗？是否有些实数不是代数方程的解？

1740年，莱昂哈德·欧拉（1707—1783，一位瑞士出生的孜孜不倦的数学家，其名字的谐音是“油壶”[1]）猜想，非代数数确实存在，他称它们为超越数，因为它们超越了代数。证明超越数存在是艰难的，你如何证明一个特定的数不是一些极其冗长并且无比繁杂的代数方程的解？

超越数的存在一直是一个未解决的问题，直到1844年，法国数学家约瑟夫·刘维尔（1809—1882）想出了一个容易研究的数，并且成功证明了它不是代数数。刘维尔所选数的小数点后30位是：

.110001000000000000000001000000…

但是这个片段并不能完全揭示它的完整形式，刘维尔利用阶乘构造了这个奇特的数。一个数的阶乘是小于等于这个数的所有正整数的乘积，用感叹符号来表示：

1!=1

2!=1x2=2

3!=1x2x3=6

4!=1x2x3x4=24

5!=1x2x3x4x5=120

等等。刘维尔数（通常这样称呼它）在小数点后第1，2，6，24，120，

…位为1，其余位置为0。刘维尔设计了这样一个数，来证明它不是任何代数方程的解。越来越稀疏的非零数字是这个证明[2]的关键。

1882年，德国数学家费迪南德·林德曼（1852—1939）证明了长久以来最著名的一个无理数也是超越数，这个数就是π，即圆的周长与直径的比：

$$\pi=3.14159265358979323846264338327\,95\ldots$$

林德曼证明了π不是代数方程的解，这个事实为一个古老的难题提供了新的视角：在过去的两千多年来，数学家和非数学家都在尝试解决的“化圆为方”问题。这个问题可以简单叙述为：给出一个圆，用直尺和圆规构建一个与圆面积相等的正方形。（一个类似的难题称为“圆的矫正”，它需要构建一条与圆的周长相等的直线。）人们是如此疯狂地尝试解决这个问题，以至于古希腊语中都有了专门表示这一活动的词 τετραγωνίζειν，从字面上看，它的意思是“四角化”。[3]

用直尺和圆规构建一个几何图形与求解某些特定形式的代数方程是等价的。因为π不是任何一个代数方程的解，所以你不能在一个几何构造中表示这个数。这就意味着，用直尺和圆规构建一个与圆面积相等的正方形是不可实现的。

另一个著名的超越数用符号e表示（代表欧拉）。如果计算

$$\left(1+\frac{1}{N}\right)^{N}$$

那么在*N*趋于无穷大的情况下，结果会趋近e：

$$e=2.71828182845904523536028747135\,27\ldots$$

你也可以用下面这个包含阶乘的无限数列计算e：

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\frac{1}{5!}+\cdots$$

你可以计算它，但它不是任何一个代数方程的解。

在过去的这个世纪中，许多数已经被证明是超越数了，但是仍然没有一种通用方法来证明一个数是不是超越数。例如，对于下面这个数仍然没有结论：

$$\pi=^{\pi}$$

图灵论文（以及本书）将数限定在了实数（非虚数）。下面的图汇总了实数领域内几个最重要的类别。



这个图没有按比例来画。

等等，这么说是什么意思？

这些类别中的数都有无穷多个，不是吗？无穷多个整数，无穷多个有理数，无穷个无理数，不是吗？无穷是无穷的，不是吗？没有不同大小的无穷，不是吗？不存在一个无穷大于另一个无穷，不是吗？

对吗？

不论我们是从哲学、神学还是数学哪个方面谈及无穷，它永远都不是一个简单的话题。然而在数学中，无穷几乎不可回避，我们不得不鼓起所有勇气去研究无穷这个概念。

自然数的无限增大似乎是无穷大这个概念的根源。无论我们数到哪个数，总能再多数一个。实数当然也是可以无限增大的，但那只是因为它们会跟着自然数一起增加。当我们一次又一次地细分连续统时，我们便开始思考实数的无穷小了。

这两个无穷——无穷无尽的自然数和无穷稠密的连续统——在某些方面有相似之处吗？或者说它们完全不同？

如果我们掌握了集合论的一些基本知识，接下来的讨论就会简单些了。集合是由一些称作集合元素的对象组成的。集合通常用大括号表示，例如，

$$\{1,2,3,4\}$$

是前四个自然数的集合。集合里的元素是唯一的，比如不允许出现两个4。集合里元素的排列顺序无关紧要，集合

$$\{4,1,3,2\}$$

与前一个集合是一样的。集合中元素的个数称作基数，也叫势。上面的有限集合的基数是4。具有相同基数的集合称作等势的集合。

有些集合的势是有限的，有些集合的势是无限的。正整数集合：

$$\{1,2,3,\ldots\}$$

的势显然是无限的。正偶数集合的势也是无限的：

$$\{2,4,6,\ldots\}$$

这两个集合的势之间有什么关系呢？

我们也许会脱口而出，第一个集合的元素个数是第二个集合的两倍，因为第二个集合少了所有的奇数。这当然只是片面的看法。如果这

两个集合都是有限的，那么这种看法确定是对的。但是，这两个集合都是无限的，我们怎么能说一个集合是另一个集合的两倍呢？

我们来数一下第二个集合的元素。什么叫做“数”？它的意思是将这些元素与我们数数时心中默念的自然数“1，2，3，…”一一对应起来。

我们可以通过与自然数做一一对应，来数无限集合里的正偶数：

1	2	3	4	5	6	7	8	…
↕	↕	↕	↕	↕	↕	↕	↕	
2	4	6	8	10	12	14	16	…

对于每一个正整数，都有一个偶数与之对应。对于任何一个偶数，都有一个正整数与之对应。这么一看，这两个集合现在似乎变得一样大了，也就是说它们是等势的。这是怎么回事？（事实上，无限集合的这种独有特征是伽利略在1683年[4]提出的，因此有时也称作伽利略悖论。）

似乎没有人过多关注过这个悖论，直到格奥尔格·康托尔（1845—1918）跟它较起劲来。康托尔，伟大的数学家，出生于圣彼得堡，以建立集合论而闻名。他的父亲是一名商人，在各方面引导儿子出类拔萃，他的母亲出身于博姆音乐世家。康托尔崭露了自己在艺术和音乐上的天赋，但是他在17岁时决定“献身于数学”。[5]他去了苏黎世理工学院和柏林大学。1869年，康托尔在哈雷大学获得了一份教学工作，并在那里度过了余生。

在1873年寄给数学家理查德·戴德金（1831—1916）的一封信中，康托尔探索了类似自然数与偶数之间的对应，并且考虑是否可以在自然数和实数之间建立类似的对应。他怀疑这不可能，但是无法解释这是为什么。“我找不到我要寻找的答案，也许它很简单。”康托尔写道，[6]这是他著名的遗言。

如果集合中的元素能与自然数一一对应，那么我们称这个集合为可数的。如果我们能将集合中的元素按照某种方式排序或列举出来，那么这个集合就是可数的，因为任何一个列表都是可以标号的，也就是将各项与自然数1，2，3，…一一配对。所有有限集合当然都是可数的。真正的难题来自于无限集合。

例如考虑由全体整数构成的集合，其中包含正数、负数和零。这个集合是可数的吗？是的。因为我们可以从零开始列举所有这些整数：

0
1
-1
2

-2
3
-3
…

这不是列举整数的通常方法，但是它向我们展示了，单个列表能包含所有的整数。

有趣的是，有理数也是可数的。我们从正有理数开始，而且不要担心数列里有一些重复的数：

1/1
1/2
2/1
1/3
2/2
3/1
1/4
2/3
3/2
4/1
…

看出规律了吗？数列中第一项的分子分母之和是2，接下来两项的分子分母之和是3，之后三项的分子分母之和是4，以此类推。这个列表就包含了所有的正有理数。只需一正一负地交替列举，我们便能把负有理数也加进来。因此，有理数是可数的。

在1874年发表的一篇论文“关于实代数数集合的性质”[7]中，康托尔指出甚至代数数都是可数的。正如我们知道的，代数数是代数方程的解，代数方程的一般式是

$$a_N x^N + a_{N-1} x^{N-1} + \ldots + a_2 x^2 + a_1 x + a_0 = 0$$

其中N是正整数，a_i是整数。对于任何一个代数方程，将所有的系数（a_i的值）和N相加，我们称所得的值为方程的高。对于某个特定的高（例如5），存在有限个数的方程，每个方程至多有N个解。所以，所有的代数数都可以根据它的高和解来排列。因此，代数数是可数的。

那么超越数呢？超越数是否可以按照某种方式列成一张表？这看上去极不可能！我们甚至没有检测一个特定的数是否是超越数的一般步骤！

那么包含了代数数和超越数的实数呢？实数可数吗？

在1874年康托尔证明代数数可数的同一篇论文中，他也证明了实数是不可数的。

康托尔首先假设实数是可数的。他假设存在一种枚举实数的方法，并且这些实数已经按照这种方式排列好了，我们用带下标的希腊字母ω来标记：

$$\omega_1\ \omega_2\ \omega_3\ \omega_4\ \omega_5\ \omega_6\ \ldots$$

康托尔打算证明这个列表是不完整的——无论怎样构造这个列表，它都不可能包含所有的实数。

随便选择一个数α和一个稍大的数β。你可以像这样在数轴上表示这两个数：



现在，从左至右依次查看那个列表里的数，找出两个大小在α和β之间的实数，这两个数均大于α并且小于β。我们称这两个数中小的那个为α'，大一些的为：β'



从刚才停下来的地方开始，继续往下搜索列表，直到碰上两个新的大小介于α'和β'之间的数，称这两个数为α''和β''



继续：



这个过程显然可以无限进行下去。你总能在刚才的两个数之间再找到两个新的数。

你怎么知道的？很简单。假设你被卡在了这一步



上标(ν)表明有ν个小撇，也许是一亿亿亿个，但仍是个有限的数。现在，无论如何继续在枚举好的实数列表中搜索，都不能在$\alpha^{(\nu)}$和$\beta^{(\nu)}$之间找到另一组数。很显然，你的实数列表是不全的。这个列表缺少了$\alpha^{(\nu)}$和$\beta^{(\nu)}$之间的每一个数。例如，夹在$\alpha^{(\nu)}$和$\beta^{(\nu)}$正中间的数是这两个数的平均数，也即：

$$\frac{\alpha^{(\nu)}+\beta^{(\nu)}}{2}$$

这还只是个开始。你的数列漏掉了许多数。

这就是你知道这个过程必定会无限进行下去的原因。所有的α会持续增大，而β会持续减小，但是最大的α不能大于最小的β。（当你在最后一组α和β之间找到两个新数的时候，那个小的数总是α，大的数总是β。）α和β都有一个界线（极限），康托尔用无穷符号作为上标来标记：α^{∞}和β^{∞}。

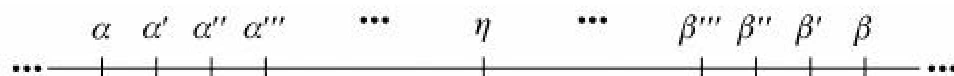
α^{∞}是否可能小于β^{∞}？我们来看一下：



不，这不可能。如果α永远不会大于α^{∞}并且β永远不会小于β^{∞}，那么这个实数数列就会丢失每一个在α^{∞}和β^{∞}之间的数，第一个能想到的就是：

$$\frac{\alpha^{\infty}+\beta^{\infty}}{2}$$

α^{∞}一定等于β^{∞}。康托尔称这个极限为η（希腊字母beta）：



因为这是一个永远持续的过程（我们已经说明了它不可能在某一点停下来），α永远无法达到η，β也如此。现在，你知道这是什么意思了，对吗？这意味着，η不在最初的那个实数列表里！

如果η真的在这个实数数列里，那么它本该在某一次搜索下一个α和β的时候出现，但考虑数列中搜索到η之前的那一对α和β：

$$\cdots\ \alpha\ \ \alpha'\ \ \alpha''\ \ \alpha'''\ \cdots\ \alpha^{(v)}\ \ \eta\ \ \beta^{(v)}\ \cdots\ \beta'''\ \ \beta''\ \ \beta'\ \ \beta\ \cdots$$

现在这个实数列表里漏掉了在$\alpha^{(v)}$与$\beta^{(v)}$之间除了η以外的每一个数。

我们考虑完了所有情形。没有一个成立，没有一个符合逻辑，这都怪我们最原始假设是错的——我们假设了实数是可枚举的，这一切一定都是因为我们根本做不到这一点。

整数是可数的，有理数是可数的，甚至代数数都是可数的。然而，实数都是不可数的。

康托尔考虑将实数不可数的性质作为超越数存在的新证据。（如果超越数不存在，那么实数就等同于代数数，从而可数。）康托尔最终意识到至少有两种无穷：可数的无穷和不可数的无穷，即自然数的无穷和连续统的无穷。自然数、有理数，甚至代数数的无限集合都是可数的。当我们将超越数放进来的时候，我们突然间就置身在一个完全不同的世界中了。我们眼前有两种不同的无穷的势：一种势适用于自然数、有理数与代数数；另一种势适用于实数和连续统。

康托尔的成果在当时饱受争议，他自己也一直没能摆脱这种争议。自康托尔后，没有哪个数学家再像他那样思考无穷了。可数的无穷和不可数的无穷之间的区别已被证明是极其有用的，即使想象一种简单的无穷就足以震撼人心。

有一种流行的说法是，对无穷的冥思苦想让康托尔本人最后疯掉了。康托尔确实在他生命的最后20年中经常出入精神病院，但这也许是一种跟职业无关的躁郁症。[8]然而，最糟糕的是，疲劳和压力经常使他的精神疾病发作，而且这种压力与其非传统的数学理论是否被接受息息相关。在休养期间，他的兴趣已不在数学上了。他研究过哲学、神学、形而上学，以及“培根是莎士比亚戏剧的真正作者”这一假说。

有限集合与无限集合有很多不同，其中一个很大的不同就是真子集，也就是那些与集合自身不相同的子集。有限集合的真子集总是有较小的基数，这一点显而易见。无限集合的真子集也可以有较小的基数。（例如，自然数集就是实数集的真子集，它们的基数是不同的。）然而在有些情况下，有些集合的真子集有着与集合本身一样的基数。这只对无限集合成立。自然数集是整数集的真子集，整数集是有理数的真子集，有理数又是代数数集的真子集。所有这些无限集合都有相同的基数，它们是等势的。

实数的各种真子集也有可能是相互等势的。想想1与0之间的实数。这些数可以与大于1的实数一一对应，只要把每个数都用1除一下就好

了。例如，0.5对应2，0.25对应4，0.1对应10，0.0001对应10000。这一事实非常有用，意味着我们可以考察0和1之间的实数的某些性质，其结论将适用于所有的实数。（图灵在他的论文中运用到了这一概念，康托尔也用到了它。）

康托尔在探索无限集合时还有其他惊人发现。他发现我们可以在连续统（直线上的实数）和平面上的点，乃至*N*维空间中的点之间建立一一对应关系。

下面我们只看x和y坐标都在0和1之间的那部分平面区域。平面上的任何一点都可以表示为数字对(x，y)，并且这两个数中的任何一个数小数点后都有无穷位。在下面的表示中，x的小数点后每一位都用带有下标的a来表示：

$$x=.a_1a_2a_3a_4\ldots$$

y同样如此：

$$y=.b_1b_2b_3b_4\ldots$$

现在，把这些数插在一起，形成一个新的数：

$$.a_1b_1a_2ba_2a_3b_3a_4b_4\ldots$$

这是由两个实数压在一起形成的一个实数。每一个二维的点都对应着连续统上的一个实数。因此，平面上的所有点和直线上的实数有着一样的基数。康托尔被这个发现震撼得说不出话来。他在给戴德金[9]的信中写道：“Je le vois, mais je ne le crois pas.”（我了解它，但我不相信它。）

1891年，康托尔发表了另一个实数不可数的证明，[10]从那以后，这个证明至今令人拍案叫绝。康托尔的证明涉及了集合而非数字，并且比我即将展示的例子更具一般性，二者思路是一样的。这种思路被称作对角线证明法（diagonal proof）、对角线过程（diagonal process）、对角线论证（diagonal argument）或者对角化（diagonalization），原因大家马上就知道了。无论怎么称呼它，总少不了对角线一词。

来看0和1之间的实数。假设我们设计了一种列出所有实数的方法。（正如你所料，这是另一个反证法。）假设这个排列像下面这样：

.1234567890…
.2500000000…
.3333333333…
.3141592653…
.0101101110…
.4857290283…
.0000000000…
.9999999999…
.7788778812…

.2718281828…

…

我们似乎有了一个良好的开端。这个数列包括0、1/4、1/3、π/10、e/10，还有之前那个连续数字“1”越来越多的无理数，以及一些不大能辨认出来的数。每个数都有无限的十进制小数位（即使它们都是0），并且这个列表有无限多个数。

即便这个列表是无限的，我们仍然可以说服自己这里面漏掉了一些东西。我们从左上角到右下角看这个列表中的数字，这些数字用粗体显示：

.1234567890…
.2500000000…
.3333333333…
.3141592653…
.0101101110…
.4857290283…
.0000000000…
.9999999999…
.7788778812…
.2718281828…
…

现在用这些粗体数字构建一个数：

.1531190918…

因为实数列表是无限的，每个数的位数也是无限的，所以上面这个数有无限位。现在将这个数的每一位都增加1。如果这一位是9，就把它变成0：

.2642201029…

这个新数是否在原来的列表里？让我们一步一步来看：这个新数是否是列表中的第一个数？不是，因为列表里第一个数的第一个位是1，而这个新数的第一位是2。

这个数是列表中的第二个数吗？也不是，因为数列中第二个数的第二位是5，而新数的第二位是6。

这个数是数列中的第三个数吗？不是，因为数列中第三个数的第三位是3，而新数的第三位是4。

等等等等。这个新数不是列表里的第N个数，因为第N个数的第N位与这个新数的第N位不相等。

因此，这个列表是不全的，我们先前的假设有问题。枚举0和1之间的所有实数是不可能的，我们再次看到实数是不可数的。

当我们对代数数进行同样的操作会发生什么？我们已经知道如何枚举代数数，这不是问题。当构建一条对角线并且改变所有位上的数字时，所生成的数并不在这个列表中。这就意味着生成的数不是代数数，而是超越数。

你可以将代数数按照多种不同的方式排列，你可以制定不同的规则让对角线和原数列中的每一个数都不同。每次这么做，你都将得到一个新的超越数。

1895年，康托尔选择用希伯来文字母表中的第一个字母加上下标0（$\aleph_0$，读作“阿列夫零”）来表示可数的自然数集合（因此也是任何可数的无限集合）的基数。康托尔称这是第一超限数。他将它与其他超限数（$\aleph_1$、$\aleph_2$、$\aleph_3$等）结合起来建立了超限数的整个数学体系。

如果可数集合的基数是$\aleph_0$，那么实数的不可数集合的基数是什么？我们能否表示这个基数？

也许吧。我们先来看一个有限的集合，它仅有3个元素：

$$\{a, b, c\}$$

这个集合有很多子集，你能构造出多少个来？（一个集合的所有子集的集合叫做幂集。）你可以动手尝试一下，但是不要忘了空集和包含所有3个元素的集合：

$$\begin{array}{ll} \{\ \} & \{a, b\} \\ \{a\} & \{a, c\} \\ \{b\} & \{b, c\} \\ \{c\} & \{a, b, c\} \end{array}$$

含3个元素的集合共有8个子集，而且这并非巧合：

$$2^3=8$$

其中指数是原集合元素的个数，结果是子集的个数。一个含4个元素的集合有16（2的4次方）个子集，含5个元素的集合有32个子集。

为了更好地揭示这种联系，我们还有一个更具条理的方法来枚举这

些子集。画一个表格原集合中的每个元素各占一列，用0和1来表示这些元素是否在各个子集中：

a	b	c	子集
0	0	0	$\{\ \}$
0	0	1	$\{c\}$
0	1	0	$\{b\}$
0	1	1	$\{b, c\}$
1	0	0	$\{a\}$
1	0	1	$\{a, c\}$
1	1	0	$\{a, b\}$
1	1	1	$\{a, b, c\}$

这些三位的0和1组合依次与二进制数的0到7相对应。3位得到8个二进制数。一般的规则是：

$$\text{幂集的势} = 2^{\text{原集合的势}}$$

一个含10个元素的集合，其幂集含有1024个元素，一个含100个元素的集合，其幂集含有1267650600228229401496703205376个元素。

再来关注自然数（因为需要，把0也算进来了）：

$$\{0, 1, 2, 3, 4, 5, \ldots\}$$

这个集合的基数是 $\aleph_0$。它有多少个子集呢？或者说，它的幂集的基数是多少？它的基数是：

$$2^{\aleph_0}$$

我们有必要进行进一步的确认。我们来构建一个与类似有限集合的表格（显然设法画全）。各列的第一行依次是自然数集合里的所有元素。每一列的0和1表示该元素在哪些子集中，所得的子集写在每行最右边：

0 1 2 3 4 5 … 子集

0 0 0 0 0 0 … { }
1 0 0 0 0 0 … {0}
0 1 0 0 0 0 … {1}
1 1 0 0 0 0 … {0，1}
0 0 1 0 0 0 … {2}
1 0 1 0 0 0 … {0, 2}
0 1 1 0 0 0 … {1, 2}
1 1 1 0 0 0 … {0, 1, 2}
… … … … … … … …

在这里，我们其实是在尝试枚举所有无限多种可能的0和1的组合。我们在该列表的每串数字前加一个小数点：

.000000…
.100000…
.010000…
.110000…
.001000…
.101000…
.011000…
.111000…
…

它们都是0和1之间的二进制数，并且也是0和1之间的所有二进制数，因此也就是0和1之间所有的实数。[11]我在之前展示了如何将0和1之间的实数与整个实数一一对应，这就是说实数可以与自然数的幂集中的成员建立一一对应关系。这个幂集因此有和连续统一样的基数。

因此，连续统的基数是：

$$2^{\aleph_0}$$

其中 $\aleph_0$是自然数的基数。

康托尔证明了将任何一个非空集合的元素与其幂集的元素一一对应是不可能的，这个事实对于有限集合很明显，但对于无限集合就不明显了。这个结论现在称为康托尔定理，它也是1891年那篇介绍对角化技巧的论文的主要成果。正如一个集合有幂集一样，一个幂集同样可以有自己的幂集，等等。所有这些集合都有不同的基数。

康托尔推测，连续统的基数是 $\aleph_0$之后的下一个超限数，他称这个基数是 $\aleph_1$。这个推测称为康托尔的连续统假设，用数学语言表示为：

$$\aleph_1 = 2^{\aleph_0}$$

康托尔不懈地证明他的假设，但是始终没有成功。问题在于，在 $\aleph_0$ 和连续统的基数之间可能存在一些超限数。

尽管如此，这一切的深刻涵义在于可数集合的基数不仅仅是比连续统的基数小，

$$\aleph_0 < 2^{\aleph_0}$$

而且是非常，非常，非常，非常，非常小：

$$\aleph_0 <<<<<<<<<<<<<<<<<<<<<<<<<<<<< 2^{\aleph_0}$$

事实上，连续统与可数集的唯一区别在于，是否包含超越数。我们不得不承认，超越数，那些1844年之前甚至还不能证明其存在性的数，其实占了实数的绝大部分。事实上，它们几乎占满了所有的实数。

千百年来，我们对于数的概念完全是偏颇和扭曲的。我们总是重视整洁、秩序、模式，然而我们又生活在一个折中与近似的世界中。我们只关注那些对我们有意义的数字。为了数农场里的动物，我们发明了自然数；为了测量，我们发明了有理数；而在高等数学中，我们又发明了代数数。我们从连续统中挖出了所有这些数，却完全无视了实数海洋中其他有如微生物一般繁多的数。我们活在一种很安逸的幻觉中：有理数比无理数多得多，代数数比超越数多得多，当然这仅仅是我们的一厢情愿。然而事实上，在连续统的世界中几乎每一个数都是超越数。

这些超越数到底是什么？它们中的大多数仅仅就是随机的数字序列，完全没有模式、规则和意义可言。实际上，任何一个随机的数字序列几乎都是超越数。

我们向标靶投掷一个飞镖，然后使用一个带渐进高倍镜的无比精确的尺子测量飞镖和靶心的距离。我们首先量出整的英寸数，然后量十分之一英寸，再后是百分之一英寸，如此可以一直量下去，永无止息。所得结果是有理数（比如正好1.437英寸）的概率基本可以忽略不计。

当然，真正测量飞镖的时候我们不得不面对真实世界的情况。飞镖总不可能正好把一个原子劈开吧！显然，飞镖会插进标靶软木离散的分子之间，而且当我们真的放大到了分子世界时，会发现分子震荡得太快难以精确测量，而且光的波长有限会引起失真，海森堡不确定性原理又会在某个时候横插一脚，所以我们什么都不能真正确定下来。

在这样的微观世界中，连续统的概念是如此地离奇而让人绝望。而当我们把目光从分子转移到宏观宇宙中，同样也会产生疑问，无穷在现实世界中到底存在不存在。尤其是考虑到，宇宙大爆炸很可能发生在有限的时间以前，只释放了有限的物质和能量，因而宇宙似乎应该由离散而并非连续的结构来刻画。

我们还想知道：康托尔对于可数集合和不可数集合的探索仅仅是高度抽象（甚至还值得怀疑）的理论数学呢？还是可以在现实生活中有其用武之地呢？

尽管在现实世界中很难找到无限的事物，但是无限的概念在数学上还是有用的。后来发现，某些有现实意义的数学证明，包括图灵论文里的证明，核心问题就在于可数集合与不可数集合的区别上，如下图所示。



看到问题所在了吗？

---

[1] “油壶” 的英文是 “oiler”，与欧拉名字 “Euler” 发音相似。——编者注

[2] 这个证明在Edward B. Burger和Robert Tubbs的著作*Making Transcendence Transparent: An Intuitive Approach to Classical Transcendental Number Theory*（Springer，2004），9-26中涉及。

[3] 出自E.W. Hobson的*Squaring the Circle: A History of the Problem*（Cambridge University Press，1913），3。

[4] 伽利略·伽利莱，*Two New Sciences* （《两种新科学》），2E，Stillman Drake译（Wall & Emerson，2000），40-41。翻译基于 *Opere di Galileo Galilei*（Florence，1898），VIII，78-79。显然，伽利略不是第一个发现这个悖论的人。至于其他人，见斯蒂芬·科尔·克莱尼的《数学的逻辑》（*Mathematical Logic*，Wiley，1967，Dover，2002），

pg.176，脚注121。
[5] 约瑟夫·沃伦·道本，*Georg Cantor*：*His Mathematics and Philosophy of the Infinite*（Harvard University press，1979，Princeton University Press，1990），277。
[6] 奥尔格·康托尔写于1873年11月29日的书信，来自 *From Kant to Hilbert:A Sonrce Book in the Foundations of Mathematics*（Oxford University Press，1996），Vol. II，844。
[7] 可从 *From Kant to Hilbert*，Vol. Ⅱ，839-843查到。
[8] 道本，*Georg Cantor*，285。
[9] 1877年6月29日的信，*From Kant to Hilbert*，Vol. Ⅱ，860。
[10] 格奥尔格·康托尔，“关于流体理论的一个基本问题”，*From Kant to Hilbert*，Vol. Ⅱ，920-922。
[11] 看起来，我们似乎无意中发现了一种枚举0和1之间所有实数的方法。这种方法很明显：小数点后的第一位在0和1之间交替变化，同时第二位以一半的速度交替变化，以此类推。我们可以轻易地延续这个列表。然而，问题在于这个数列永远不会包含超越数。这个列表中的所有数小数点后都只有有限个非零数。

第3章
几个世纪以来的发展

随着1999年12月31日午夜倒计时几秒的来临，伴随新年而来的庆典也沉浸在焦虑和担忧之中。在午夜的钟声里，我们有可能见证一场波及全世界互连计算机系统的重大技术事故及宕机，有些人甚至认为这是无法避免的。这场危机不是全球恐怖主义行为，而是由计算机程序员使用了近半个世纪的小捷径造成的。在众多系统的各种应用程序中，人们使用年份的后两位数字表示年份，例如用75代替1975，以此来节省宝贵的计算机存储空间。在那个午夜，这个两位数字的年份将会从99变成00，突然变得很小，而不是更大了。一个曾经无害的捷径竟然变成了一个危险的错误，这个故障有个高科技别名，叫作Y2K（千年虫）。

当然，程序员们早在几十年前就知道这个即将发生的问题。从1998年起，众多耸人听闻的书名开始进入公众的视野，例如*Y2K: The Day the World Shut Down*、*Deadline Y2K*、*Y2K: It's Already Too Late*、*Y2K: An Action Plan to Protect Yourself, Your Family, Your Assets, and Your Community on January 1, 2000*、*101 Ways to Survive the Y2K Crisis*、*Y2K for Women: How to Protect Your Home and Family in the Coming Crisis*、*Crisis Investing for the Year 2000: How to Profit from the Coming Y2K computer Crash*、*Y2K: A Reasoned Response to Mass Hysteria*、*Spiritual Survival During the Y2K Crisis*、*Y2K: The Millennium Bug – A Balanced Christian Response*、*Awakening: The Upside of Y2K*，以及针对儿童阅读的*Y2K-9: The Dog Who Saved the World*。不久，电视专题节目和杂志也加入这一行列。1999年4月的《连线》杂志刊出一个不祥的黑色封面，上面用大字写着"Lights Out"（熄灯），用小字写着"Learning to Love Y2K"（学会去爱千年虫）。[1]

我们被告之，计算机已经装进了几乎所有的电子产品中，因而届时会有大到大面积停水停电、飞机坠落、汽车失控，小到微波炉罢工、录像机出错等种种事故发生。

20世纪是一个众多科学和技术都在发展进步的时代，但是现在，这个技术却要给我们的洋洋自得一个狠狠的教训。

之前的世纪之交从未有过这样的恐惧。19世纪也是一个科技飞速进步的时代，但没有什么会在午夜时分毁掉，新的世纪充满光明。科学家们似乎已接近了解了所有的知识，当时很多著名的物理学家，比如开尔文勋爵(1824—1907），成功预言了物理世界中最后几个谜题也会很快被解决，这其中包括了以太能填充整个空间并为光线及其他电磁辐射的传播提供介质的本质原因。

数学，也许是19世纪中最接近计算机科学的学科，也有很大进展，而且远超预期。漂亮地逃过了一场危机之后，数学似乎变得前所未有的强大。

19世纪的这个数学危机涉及的领域可以追溯到约公元前300年，即欧几里得几何。（欧拉名字的发音是oiler，欧几里得的发音是yoo-clid。）

欧几里得的《几何原本》以一系列定义开始，随后是五个公设和一些常用概念（也称为公理）。通过这几个基本假定，欧几里得推导出了数百个定理。

欧几里得的前四个公设如此显然，看上去几乎没有写下来的必要。前三个公设说的是可以用直尺和圆规画直线和圆，第四个同样简单。

1. 从一点到另一点可以画一条线。

2. 直线可以向两边不断延伸。

3. 给定圆心和半径可以确定一个圆。

4. 所有的直角都相等。

这四条公设简洁，明了，不言自明。相比之下，第五个公设可谓是出了名的冗长与晦涩。

5. 如果一条直线与另两条直线相交，且在同一侧形成的两个内角之和小于两直角，那么无限延长这两条直线，它们会在这一侧相交。[2]

这个公设定义了直线不平行的条件。

从最早对《几何原本》的评论开始，第五个公设就颇有争议。一些数学家认为，第五公设是多余的，是不必要的，其实是可以从前四个公设推导出来的。但是，所有企图推导出第五公设的人都失败了。那些成功了的人，全都不知情地暗中用到了与第五公设等价的假设。

在19世纪初，一些数学家开始探索另一种方法：假定做出了一些违背第五公设的假设。也许无论它们与其他直线相交的角度是多少，两条直线总是相交的。也许两条直线永不相交。如果欧几里得的第五公设真的是多余的，那么矛盾总会在某个地方出现，我们就相当于用反证法证明了第五公设。

然而，事情并没有那样发展下去。在德国、匈牙利和俄罗斯，卡尔·弗里德里希·高斯（1777—1855）、约翰·波尔约（1802—1860）和尼古

拉·伊万诺维奇·罗巴切夫斯基（1792—1856）都独立地发现，修改第五公设并不会导致矛盾，而是创造了一个虽然奇怪但前后完全一致的新几何世界产物。

在一个不太成熟的时代，数学家可能会摒弃这些令人厌恶的非欧几何，或者因基本几何被这些荒谬的构造弄得站不住脚而绝望。但是他们没有那样，数学家接受了这些欧氏几何的替代物，学到了有关数学本质的重要一课。

欧几里得一直是对的，第五公设应该纳入他的基本假设当中。作为平面的几何学，欧氏几何的第五公设是必需的，但它并不是唯一的可能。用一些其他公设替换它，会产生与欧氏几何一样合理和有趣（甚至更有趣）的几何学。这些非欧几何与我们称之的“现实世界”有什么关系吗？当然有时候会有。一种非欧几何描述的是球体的表面，从某些角度来看，球面比平面更贴近“现实世界”。

19世纪的数学家对欧几里得在《几何原本》中使用过的公理化方法，也有了更加新颖深刻的理解。（虽然欧几里得和亚里士多德区分了公设和公理[3]，但是这种差别在近代很大程度上已经消失了。）一个数学系统首先要从一些特定的公理开始，然后不断证明这些公理的推论并一直继续下去。根据我们的感受，这些公理可能与我们对现实世界的直觉看法相符，也可能不相符。两千年来，模仿真实世界的想法事实上束缚了几何学的发展。如果公理可以摆脱现实世界，变得足够抽象，那么数学自身就解放了，能够探索一番新的前景。我们必须抽象地看待数学公理和由此产生的数学体系，心里不能带有任何假设。

或者如同一位年轻的数学讲师曾经深思的：“我们必定可以用桌子、椅子和啤酒杯来代替点、线和面。”[4]

这位年轻的数学讲师就是大卫·希尔伯特（1862—1943），未来最杰出的数学家之一。希尔伯特的出生地很靠近哥尼斯堡，一个波罗的海港口城市，当时是东普鲁士的首都。在希尔伯特出生前，哥尼斯堡就是一个数学胜地，城市中交错在普莱格尔河上的七座桥就是欧拉解决了的拓扑谜题的原型。

哥尼斯堡还是哥尼斯堡大学的所在地，哲学家伊曼努尔·康德（1724—1804）曾在那里研究和授课。希尔伯特也进过这所大学，并在那里短暂地教课。1895年，希尔伯特接受了哥廷根大学的一个职位。他第一次访问哥廷根是在9年前：“他发现自己被这个小镇和美丽多山的乡村吸引了，这座小镇与喧闹的哥尼斯堡城和城外那平坦的草原都是如此的不同。”[5]

在希尔伯特到来之前，哥廷根大学就已闻名于世。1833年，高斯和物理学家威廉·韦伯（1804—1891）在那里合作发明了电磁电报。随着

希尔伯特和菲利克斯·克莱因（1849—1925）对数学系的管理，哥廷根即将成为全世界数学家的圣地。

早些年，希尔伯特解决了代数不变量和数域领域的未解问题，并借此赢得了声誉，但是在1898～1899学年中，他的兴趣有了不寻常的改变。希尔伯特教授几何课，而这门课通常不在大学教授，学生们已经在基础教育阶段完成了欧几里得基础课程。

希尔伯特的几何课教授模式与欧几里得几何相似，从几个公理（事实上是几组公理）出发，然后从这些公理中推导出许多定理。不同的是，希尔伯特的推导无比严谨。希尔伯特重新思考了几何学，并且重新进行了公理化。这是将非欧几何知识融入其自身概念的新时代欧氏几何。1899年，希尔伯特出版了他的几何学讲义《几何基础》（*Grundlagen der Geometrie*），这本书立即成为了一部数学经典。（英文第2版翻译自德文第10版，由Open Court出版社于1971年出版，目前仍在重印）。

希尔伯特没有像欧几里得那样将他的几何书称为《几何原本》，而是叫做《几何基础》。对于希尔伯特来说，将几何学建立在稳固的公理体系根基上，比解决或证明定理重要得多。建立几何根基的关键工作是证明公理体系的一致性，也就是这些公理不会引起任何矛盾。为此，希尔伯特把他的几何类比到了实数平面上。这基本上就是笛卡儿坐标系上的解析几何。于是希尔伯特几何学的一致性问题就转化成了实数算术的一致性问题。

在那个年代，希尔伯特并不是唯一对构建数学基础感兴趣的数学家。1889年，朱塞佩·皮亚诺（1858—1932）将公理化方法应用在了一个大多数普通人都认为没有必要的地方——自然数算术的公式化上。那时并不太有名（不过现在评价很高）的戈特洛布· 弗雷格（1848—1925），使用一种全新的记号重新构造了数理逻辑，并在其1879年出版的名为《概念文字》（*Begriffschrift*）[6]的小册子中做了描述。后来，弗雷格出版了《算术基础》（*Grundlagen der Arithmetik*，1884），试图通过数理逻辑来建立实数算术的基础。之后，弗雷格又在篇幅更大的著作中详细阐述了他的系统。他在1893年出版的《算术的基本法则》（*Grundgesetze der Arithmetik*）第一卷中，将集合论和数理逻辑结合起来用以建立实数的合理性。

世纪之交到来之前，这些数学的根基在很短时间内先后各归其位，数学看似处在顺利的轨道上。1900年8月，希尔伯特受邀在巴黎召开的第二届国际数学家大会上进行重要的演讲。考虑到这次演讲将为数学开辟一个崭新的世纪，希尔伯特不太确定要说些什么。

希尔伯特求助于他在哥尼斯堡大学的好友，出生于立陶宛的数学家

赫尔曼·闵可夫斯基（1864—1909）。闵可夫斯基建议他在演讲中尽量展望数学而不是进行回顾：

> 最吸引人的莫过于展望未来，你可以列举出一系列有意义的、值得数学家们在即将到来的新世纪为之研究奋斗的数学难题。如果你选择了这样的主题，人们在数十年后还会对你的演讲津津乐道。[7]

因此，在1900年8月8日，希尔伯特按照闵可夫斯基的建议，开始了他的演讲：

> 我们当中有谁不想揭开未来的帷幕，看一看在今后的世纪里我们这门学科发展的前景和奥秘呢？下一代的主要数学思潮将追求什么样的特殊目标？在广阔而丰富的数学思想领域里，新世纪将会揭示什么样的新方法和新成就？[8]

然后，希尔伯特大概地讨论了一些需要新世纪数学家们解决的问题。他向听众保证，这些问题终有一天会被全部解决。

> 无论这些问题看起来多么难以解决，无论我们在它们面前显得多么无助，然而我们可以肯定的是，解答它们必然要经过有限步骤的纯逻辑过程……对每个数学问题必然可解的坚定信念，恰恰强烈地激励了我们每个数学工作者。我们会听到内心深处那永恒的呼喊：这里有一个数学问题，去找出它的答案吧！你能通过纯思维找到它，因为在数学中没有不可知（ignorabimus）。[9]

尽管ignorabimus是一个生僻的拉丁词，但是希尔伯特的听众、这些数学家们还是可以很容易解析它的含义，即“我们不知道”。一些人还将这次演讲与1876年物理学家埃米尔·杜布瓦—雷蒙（1818—1896）所做的演讲关联起来。杜布瓦—雷蒙在那次演讲中悲观地总结道：“考虑到物质和力的谜一样的本质……科学家们不得不接受这个残酷的定论：它是不可知的。”[10]

对于杜布瓦—雷蒙来说，物质和能量的本质是永远无法揭开的，而乐观的希尔伯特显然不能容忍这样的态度。尤其对于数学，他很明确地说，没有什么是“我们不知道”的。

然后，希尔伯特提出23个来自于多个数学领域的未解决问题，向他的同僚们提出了挑战。（当时因为演讲时间有限，只提及了10个问题。发表出来的演讲全文中包含了所有23个问题。）在这些问题中，有些十

分深奥，而有些则是其领域里的基础问题。

第一个被提到的问题就是“康托尔的连续统基数问题”，即连续统的势是不是紧挨着自然数集的势之后的第一个超限数，或者说，这两个超限数之间是否有其他的超限数。在当时，对格奥尔格·康托尔的工作的争议已经少了很多，而希尔伯特就是康托尔最有力的拥护者之一。

第二个问题关注于“算术公理系统的相容性”。希尔伯特将自己的几何学的一致性建立在实数系统和算术的一致性之上。由此，实数也需要被公理化，并且“需要证明它们是无矛盾的，也就是说，经过有限步的逻辑运算之后不会产生任何的矛盾结果”。[11]

对于第十问题，希尔伯特用德语说道：

> Entscheidung der Lösbarkeit einer diophantischen Gleichung.

注意单词Entscheidung，它是本书中一个非常重要的词，意思是决定、判定、判断。希尔伯特第十问题的完整叙述如下所示。

> 10. 丢番图方程可解性的判定
>
> 给定一个包含任意个未知数的有理整系数不定方程，试推导一个过程，通过有限步运算判定该方程是否存在有理整数解。[12]

是的，自丢番图撰写《算术》起已经过了1650年，而数学家们仍然热衷于研究丢番图方程。有些数学家在研究特定形式的丢番图方程，而希尔伯特询问的却是通用判定过程。要注意的是，这个问题并不是要寻求一个解决所有丢番图方程的通用方法，它寻找的是可解性的判定。考虑任意一个丢番图方程，它是可解的吗？它有有理数解吗？希尔伯特要的是一个判定过程，而且他丝毫不怀疑这样的过程是存在的，唯一的问题是要找到它。

希尔伯特定义这个问题时所使用的词，为这个特殊的判定性问题和今后一些年的其他判定性问题定了基调。希尔伯特想要一个由有限步操作构成的过程。简言之，希尔伯特想找一种算法，但是这个词（无论是英语的Algorithm，还是德语的Algorithmus）在当时还未使用，最起码没有现在的含义。现在的算法一词，还是在20世纪60年代通过计算机相关著作推广开来的。[13]

在1900年的那次演讲中，希尔伯特邀请他的听众“揭开”20世纪的“帷幕”。但是无论是他自己还是其他任何人都没有想到其后所发生的事情。如果当时的物理学家们坚信，他们已经接近了解所有的知识，那么他们的希望在1905年将被彻底摧毁。那一年，如今称为物理学家阿尔伯特·爱因斯坦（1879—1955）的“奇迹之年”（annus mirabilis）。在短

短的一年中，爱因斯坦发表了一篇博士论文和其他四篇论文，建立了相对论和量子力学的基本原理。

宇宙再也不是线性的、欧氏的和确定性的了。空间与时间在相对论的宇宙中失去了依靠。在1907年著名的相对论论文中，希尔伯特的朋友赫尔曼·闵可夫斯基杜撰了时空（Zaumreit）[14]这个词。（闵可夫斯基在1902年来到哥廷根大学，但在1909年因阑尾炎猝死。）最终，在整个世纪里，量子力学中最知名的成果应该是不确定性原理（1927）。

或许正是回应了这种新的转变和不确定性，现代艺术和音乐走向了惊人的、令人振奋的方向。可视的形状和物体被肢解，然后重新组装成立体画和雕刻。随着真实世界中“客观”宇宙变得越来越不可靠，超现实主义者们转向在其潜意识生活和非理性梦想中寻求自我。

在音乐上，浪漫的时代似乎终结了。在后期，瓦格纳和德彪西等浪漫主义音乐家大量采用半音化和声的作品，不得不为伊戈尔·斯特拉文斯基充满了不和谐音及不对称节奏的《春之祭》让路。事实上，《春之祭》于1913年在巴黎首演时，听众就群情激愤，差点酿成暴动。在20世纪20年代早期，奥地利作曲家阿诺德·勋伯格提出的十二音列理论，就是对音乐和弦的根基进行重新公理化，从而创造出非欧音乐。

20世纪的数学家们也不可避免地卷入了这股颠覆性的风潮。第一个不和谐的音符在1902年被弹奏出来。

戈特洛布·弗雷格，1848年出生于德国维斯马。在希尔伯特到达哥廷根前20年，他从哥廷根大学获得了博士学位，此后任教于耶拿大学，并在那儿待了44年。其毕生之作《算术基础》的第一卷出版于1893年，他在书中尝试对所有的数学理论进行系统性的开发，第一个被选中的就是数理逻辑，现在称之为逻辑学。谁知第一卷销量太差，以至于出版商都不愿意出版第二卷了，所以1902年弗雷格试图自费出版第二卷。

与此同时，《算术基础》的第一卷引起一位重要读者的注意。

这个读者就是伯特兰·罗素（1872—1970），一个非同寻常的人物。他的第一篇数学论文发表在维多利亚执政期，而他一直活到了越南战争时期，并参加了反战运动。罗素出生于名门望族，他的祖父约翰·罗素（1792—1878）曾是英国首相，他的教父是功利主义[15]哲学家约翰·斯图亚特·密尔（1806—1873）。罗素很早就对数学产生了兴趣：

> 11岁时，我的哥哥开始教我学习欧氏几何。这是我一生中的一个大事件，就如同初恋一样使我神魂颠倒。我从来没有想到，世界上还有如此令人愉悦的东西存在。待我学完第五命题的时候，我的哥哥告诉我大家普遍都认为这个很难，但我一点也不觉得。这是我第一次意识到我可能智力非常。[16]

1902年，罗素正在忙于撰写自己的《数学的原理》（计划第二年出版），他发现了皮亚诺和弗雷格的集合论中都存在一个同样的问题。

一个集合的成员可以是其他的集合，一个集合甚至可以包含其自身。罗素于是考虑：那么由所有不包含自身的集合所组成的集合呢？它包含其自身吗？如果答案是否定的，它是一个不包含自身的集合，那么根据定义它应该包含自身；而如果它包含自身，那么它就不再是不包含自身的集合了。

这就是众所周知的罗素悖论，一个新的困扰了数学家至少两个千年的悖论。罗素后来用小镇上的理发师来类比这个问题。如果一个理发师只给所有不给自己刮胡子的人刮胡子，那么谁为他刮胡子呢？

罗素给弗雷格写了一封信，询问关于集合以所有不包含自身的集合作为成员的问题，[17]弗雷格完全被挫败了。他很快为《算术基础》的第二卷写了一个附录，不过这个问题还是没法得到解决。这个悖论成为弗雷格理论的一个基本缺陷，波及他毕生的主要成果。

伯特兰·罗素发现的这个悖论是由集合包含自身所导致的。如果去掉这种自我指涉，集合论就有可能没有悖论了。罗素开始探讨类型论，他先在《数学的原理》中谈了一些，然后在1908年的一篇论文里进行了详细的讨论。[18]罗素建立了一组具有层次结构的集合。处于结构最低层的集合称为1类型集合，它只包含个体（比如数字）。从下往上，1类型的集合只能是2类型集合的成员，2类型的集合只能是3类型集合的成员，以此类推。

罗素在1908年发表这篇论文的时候，所包含的内容远不止这点。罗素正在着手准备《数学的原理》第二卷，而他先前的导师及良师益友阿尔弗雷德·诺尔司·怀特海（1861—1947），也正在准备写他的著作《泛代数论》（*A Treatise on Universal Algebra*，1898）的第二卷。罗素和怀特海认识到他们的目标有所重合了，因此，在1906年开始合作，完成了继亚里士多德以来最重要的逻辑学著作。

由阿尔弗雷德·诺尔司·怀特海和伯特兰·罗素撰写的近两千页的《数学原理》（*Principia Mathematica*）一共三卷，分别出版于1910年、1912年和1913年。与之前的《数学原理》[艾萨克·牛顿1687年的著作《自然哲学的数学原理》（*Philosophiae Naturalis Principia Mathematica*），有时称为《数学原理》]不同的是，怀特海和罗素的著作只有书名是拉丁文。他们选用此书名也有可能受到了他们剑桥的同事乔治·爱德华·摩尔（1873—1958）的著作《伦理学原理》（*Principia Ethica*，1903）的影响。[19]怀特海、罗素和摩尔在当时都是“剑桥使徒社”的成员，这个秘密精英社团致力于研究学习哲学论文和吃光讨论会上的沙丁鱼面包。

尽管《数学原理》并不是用拉丁文写的，但它也不是完全用英文写的。书中的很多页都是密密麻麻的公式，“就像冬天早上畜棚场里母鸡在雪上留下的脚印。”一个早期读者这么说。[20]

《数学原理》综合了类型理论和数理逻辑，其中后者大都基于皮亚诺和弗雷格的理论，但使用了皮亚诺的符号表示法而不是弗雷格的图形法。《数学原理》在逻辑学上继续了弗雷格的研究，其中一个高潮部分就是怀特海和罗素证明了

$$1 + 1 = 2$$

这个式子比它看上去难多了！[21]

直到此时，大卫·希尔伯特对逻辑学还没有特别的兴趣。1904年，希尔伯特在海德堡召开的第三届国际数学家大会上做了“论逻辑和算术基础”的演讲。他在其中提及了一些可能的方法，不过直到《数学原理》出版，逻辑学的问题才走到了舞台中央。

在《数学原理》出版之后，希尔伯特和罗素本可以在数理逻辑方向进行合作，谁知国际形势如风云变幻。1914年8月4日，英国向德国宣战，德国则在几天前已向俄国和法国宣战。第一次世界大战一直持续到1918年。

罗素和希尔伯特都不是军事家。在1914年，德国政府要求主流科学家和艺术家们签署一份反驳“敌人的谎言和诽谤”的声明。希尔伯特根本不能判断德国政府所作的这些声明是否真实（政治上的判定性问题），所以他拒绝签字。[22]罗素，一位毕生反战的运动家，参加了更多的公开抗议，并因此在1916年被剑桥三一学院解聘，之后又被监禁了五个月。[23]

事实上，希尔伯特曾经在1917年邀请罗素前往哥廷根大学做讲座。即使罗素的护照没有被英国政府扣留，也很难想象这样的访问在战争期间能够成行。[24]

1917年9月11日，希尔伯特受瑞士数学家学会之邀，在苏黎世做了一次题为“公理化思考”的演讲，从而再一次公开踏入了数学基础理论领域。（尽管战争仍在进行，但因为瑞士是中立国，所以希尔伯特仍然能够在苏黎世与来自其他国家的数学家们见面。）在这次演讲中，希尔伯特第一次提出了在20世纪20年代早期开始为大家所知的希尔伯特计划，该计划偏离了逻辑主义，试图为所有的数学体系寻求严格的公理系统。为了分析公理系统，希尔伯特构思出了“元数学”和 “证明论”，能够使用数理逻辑推导其他数学体系结构中的结论。

现在，这种方法在数学中称为形式化。在希尔伯特的概念里，要构建一个形式化的数学系统，首先要构建定义、公理和从公理推导至定理的法则。理想状态下，生成的系统应该拥有以下四种互相关联的特性：

〉 独立性

〉 一致性

〉 完备性

〉 可判定性

独立性是指不存在任何冗余的公理，即没有一个公理可以由其他公理推导出来。事实上，对于独立性的思考正是数学家怀疑欧几里得五个公设的起因。他们试图利用其他四个公设推导出第五公设，由此证明欧几里得公设不符合独立性。不过后来证明，这几条公设实际上都是相互独立的。

一致性是现如今任何公理系统中最重要的特性。推导出来的任何两个定理绝不能相互矛盾！

例如，假设你设计了一个新的数学系统。这个系统包含符号、公理以及利用公理推导出定理的各种法则。事实上，这正是你主要要做的事情：利用公理推导定理，这就是所谓的证明。同时，这些法则也给出了系统内所有可能的合式公式（well-formed formula，往往简称为wff，读作woof）的语法。你可以不去推导，自己组合一个合式公式，然后再试着用公理和法则给出一个证明，说明这个公式是公理的必然推论。

我将展示一个假想数学系统中的两个合式公式。下面是第一个，我们称之为A：

gobbledygook = yadda–yadda–yadda

等号表明等式两边的表达式在某种意义下是等价的。下面是公式B：

gobbledygook ≠ yadda–yadda–yadda

除了等号替换成了不等号，这个公式和A一样。可以说，公式B是公式A的否定，或者矛盾体。

公式A和公式B是相反的，因此二者中只有一个可能为真。现在，数理逻辑中“真”的概念往往和真实生活中的一样难以捉摸。在这里，我并不想进行任何形而上学的讨论，所以仅仅粗略地将真的概念定义为“与公理所做的假设一致”。

如果你能够从已知的公理同时推导出公式A和公式B，那么该公理系统必然是不一致的，而且不仅仅是不一致——它毫无意义。这是因为，不一致性会波及整个系统，使得所有的东西同时既为真又为假，这是一个传统上称为ex falso quodlibet（从矛盾可以推出任意命题）的逻辑灾难。

这就是所谓的一致性。

完备性是指能够从已有公理推导出所有为真的公式。利用证明可以推导出为真的公式。如果通过公理既不能推导出公式A，也不能推导出公式B（即A和B都不可证明），那么该公理系统则是不完备的。事实上

究竟哪一个为真？或许你根本不知道，又或者你有一些思路，但是无论如何你不能提供一个证明。

公式为真与公式可证明之间的区别可能会令人迷惑：如果一个东西不可证明，那么我们往往无法百分之百地确定它的真假，但这并不能阻止我们在缺乏证明的情况下断言它的真假。例如，几乎所有人都相信哥德巴赫猜想为真：所有大于2的偶数都可表示成两个质数之和。然而，它仍然被称为“猜想”，因为它依然是有史以来最伟大的未被证明的数学问题之一。

（在以上讨论中，我实际上已经简化了“可证明性”和“正确性”之间的区别。可证明性是句法上的概念，它基于系统的公理以及用来推导定理的法则。而正确性则是语义上的概念，它取决于我们为系统中的符号赋予的实际含义。在本书的第三部分，我将针对这些问题进行更多的讨论。）

对希尔伯特来说，同样重要的还有可判定性。他追寻一个判定过程——用以确定任一给定合式公式的可证明性的通用方法。

如果一个数学系统被认为是不完备的，是不是也说明判定过程不存在？不一定。假设公式A和公式B都不可证明，因而系统是不完备的，但是仍然可能存在一个判定过程，通过分析公式A和B可以得出和刚才相同的结论——它们都不可证明。判定过程的存在与系统是否完备无关。

当然，更好、更强大的判定过程是那种能够确定正确性而不是可证明性的过程。即使我们不能从公理推导出来，这样的判定过程也能够直接识别出到底是A为真还是B为真。

希尔伯特在1900年的巴黎演讲中谈及丢番图方程问题时，第一次提出了判定过程的想法。在1917年关于“公理化思考”的苏黎世演讲中，希尔伯特也谈及了“有限步运算以内的数学问题的判定性问题”。在公理系统的整个数学体系中，他说，判定性问题“是流传久远的，也是讨论最多的，因为它是数学思考的本质”。[25]（当然，希尔伯特说一个东西“流传久远的，也是讨论最多的”，指的是他自己所在的数学世界的核心，也就是他自己以及他在哥廷根大学的同事和学生。在《数学原理》中，怀特海和罗素根本就不关心完备性或判定性。）

希尔伯特或许是第一个将德文单词Entscheidung（判定性）和Problem（问题）连接在一起的人，但根据记载，首次使用这个五音节合成词的是希尔伯特的一个助手海因里希·贝曼（1891—1970），那是在1921年5月10日，哥廷根数学协会的一次题为“Entscheidungsproblem und Algebra der Logik”的座谈会上。现在回顾起来，贝曼对于假想判定过程的描述真是让人瞠目结舌（黑体字原文是德文，取自贝曼文集中一

篇未发表的文档）：

> 这个问题的一个特性至关重要，就是证明过程只允许根据给定指令进行的纯机械式的计算，不允许掺杂任何严格意义上的思考活动。如果愿意，我们可以说是机械的或像机器一样的思考（说不定以后我们可以用机器来运行这种过程）。[26]

如果贝曼能寻求这种思想的逻辑性结论，那么我们今天谈论的可能就是贝曼机，而不是图灵机了！

在1922～1923学年，希尔伯特教授了一门课，叫做“数学的逻辑学基础”，也开始使用了单词Entscheidungsproblem，[27]但是直到1928年，它才开始从哥廷根扩展到数学界。同一年，希尔伯特的助手威廉·阿克曼（1896—1962）帮助希尔伯特将其课堂讲义（有些内容一直追溯到1917～1918学年）汇编成了一本小薄册子出版，名为*Grundzüge der theoretischen Logik*[28]（《数理逻辑原理》），这本书现在称为《希尔伯特与阿克曼》。

《希尔伯特与阿克曼》完全没有《数学原理》如此广阔的视野和宏伟的目标。书中只包含了除集合论和逻辑学之外的数理逻辑基础知识。但是《希尔伯特与阿克曼》以自己的方式证明了，薄薄的120页书所带来的影响实在是极其深远。书的核心部分是对于有限函数演算法（engere Funktionenkalkül）的解释，即我们今天所说的“一阶谓词逻辑”，其中涉及了对完备性和判定性的讨论。

《希尔伯特与阿克曼》的一位早斯读者是居住在维也纳的奥地利数学系学生库尔特·哥德尔（1906—1978）。关于一阶谓词逻辑，他读道：

> 在所有论域内都正确的逻辑公式是否都可以从公理推导出来，即便从这个意义上讲，公理系统是否完备还是个未解决的问题。[29]

这一段指的是这样的一阶逻辑公式，无论命题函数（现在称之为谓词）处于哪个论域，也无论对之如何解释，公式总是为真。这些所谓的“永真”公式，能不能从公理推导出来？哥德尔接受了这一挑战，并在1929年的博士论文中证明了一阶谓词逻辑在该意义下是完备的。这个定理就是哥德尔完备性定理。如果哥德尔的贡献只是止步于此，那么他或许没有今天的声望。不过对于哥德尔来说，他的探索只是刚刚开始。

一阶谓词逻辑的完备性虽在意料之中，但也是一个重要的结果。它展示了，用公理和证明机制足以推导出所有普遍成立的命题。然而，数

理逻辑显然不是无中生有。谓词逻辑的一个主要目的，就是为数字和算术建立坚实的框架和基础。为了达到这个目的，就必须将公理加入逻辑系统中，以便建立数字理论，这正是《算术原理》所追求的首要目标。那么，加入这些公理之后，是不是所有的命题或者其否定式都是可证明的呢？一阶谓词逻辑在这个更强的意义上是否也是完备的呢？这有时被称为“否定的完备性”，而证明它要难得多。这正是哥德尔紧接着要解决的一个问题。

1930年春，大卫·希尔伯特从执教工作退了下来，时年68岁。同年晚些时候，他被授予哥尼斯堡荣誉市民的称号。在“自然的逻辑和知识”会议的演讲上，希尔伯特一如既往地乐观。[30]30年前，他告诉巴黎的听众，对于数学家来说没有“我们不知道”，现在他又重申道：“对于数学家来说，不存在ignorabimus，并且据我看来，实际上自然科学中的任何分支都是这样。“希尔伯特还引用了哲学家奥古斯特·孔德的例子。孔德曾经为了说明人类的认识有局限性而声称，人类永远都不可能了解遥远的恒星是由什么物质组成的，结果几年之后这个问题就被解答了：

> 在我看来，孔德不能找出不可解问题的根本原因，在于根本就没有不可解的问题。针对愚蠢的ignorabimus，我们的答案恰恰相反：
>
> 我们必须知道。
>
> 我们必将知道。
>
> Wir müssen wissen. Wir werden wissen.

在希尔伯特被授予哥尼斯堡荣誉市民的前一天，哥德尔也到访了哥尼斯堡，出席一个数学会议。1930年9月7日，哥德尔宣称，他证明了，为了让一阶谓词逻辑能指出算术运算规则（包括加法和乘法）而必须加入的公理，将会使整个系统变得不完备。他在系统内部构造出了一个公式和它的否定式。如果算术是一致的，那么这两个公式之一必为真，但是，它们两个中的任何一个都不能被证明。

通过后来称为哥德尔配数法的方法，哥德尔使用系统中得到的算术将每一个公式和每个证明都与一个数关联起来，从而得到一个断定自己不可证明的公式。这听起来有点像说谎者悖论（“我说的每一句话都是谎话，包括这句话在内”）的数学形式，但它其实不是悖论。这个公式所断言的并不是自己为真或为假，而是自己不能被证明。如果代数系统是一致的，那么这个公式不能为假，否则会引出矛盾。因此，这个公式必须为真（但是这个真仅仅是元数学中的真，因为真假不是这个逻辑系

统本身的概念），也就是说该公式是不可证明的。

哥德尔的论文在次年发表，题为“论《数学原理》及有关系统的形式不可判定命题I”。[31]罗马数字 I 表明，哥德尔本想在这篇论文之后再接着写下去，给出更多的例子，但是该论文很快就引起了极大的反响，第二部也就没必要再写了。

不完备性定理的一个关键前提是算术的一致性。作为一个推论，哥德尔还证明了，在系统内部对算术的一致性进行证明是不可能的。因为有些公式是既不能证明又不能推翻的，所以它们有可能是不一致的。（这是不是说明了算术和初等数论是不一致的？极不可能，并且也没人这么认为。问题在于，一致性不能在系统本身中被证明。）

听说了哥德尔的不完备性定理，作为一个数学家，大卫·希尔伯特的反应有些出人意料。他“有些愤怒”，[32]但最终开始将哥德尔的发现引入自己的体系之中。

另一位数学家直接就丧失了自己对于数理逻辑的兴趣。伯特兰·罗素似乎在撰写《数学原理》中耗尽了热情：

> 最终，工作结束了，而我的智力再也没能从紧张中恢复过来。自此之后，我不能像以前那样处理困难的抽象问题了。这是我转变自己工作性质的部分原因，尽管绝不是全部原因。[33]

罗素开始追求其他爱好，比如关于哲学、政治以及社会事务的写作。1950年，他因为“在各种重要的作品中拥护人道主义理想和思想自由”[34]而获得了诺贝尔文学奖。那个时候，许多人已经忘记了他一开始是个数学家。

另一位在20世纪20年代中期居住在哥廷根的匈牙利数学家约翰·冯·诺依曼（1903—1957），也在哥德尔之后放弃了逻辑学的研究（按照他自己的说法）。不过，后来他将数理逻辑中的法则应用于对电子计算机的开发中。

哥德尔的不完备性定理并不是哥廷根所面临的最大麻烦。1933年，纳粹党禁止犹太人在德国大学中任教。对于哥廷根大学，这个几十年来以智力成就为单一评判标准的研究圣地，这个法令带来的打击是毁灭性的。理查德·柯朗（1888—1972）离开那儿去了美国，在纽约大学找到了一个职位。（今天，柯朗数学科学研究所占据了曼哈顿西第4街的一整幢大楼。）赫尔曼·韦尔（1885—1955）自己虽然不是犹太人，但是他的夫人是犹太人。像爱因斯坦一样，韦尔去了新泽西州普林斯顿高等研究院。保罗·伯尔内斯（1888—1977）被剥夺了授课的权利，但是他保住了作为希尔伯特最忠实助手的职位，一直到他离开前往苏黎世。伯

尔内斯对两卷《几何基础》（1934，1939）做出了主要贡献，尽管这两本书是以希尔伯特和伯尔内斯两个人的名字出版的。

在一次宴会上，希尔伯特发现自己坐在了教育部部长的旁边。部长问希尔伯特："没有犹太人的影响，哥廷根的数学现在怎么样了？"他回答道："哥廷根的数学？已经不存在了。"[35]

对于那些继续研究数理逻辑（哥廷根对此的影响日趋减弱）的数学家来说，那些数学问题仍然亟待解决。1928年版的《希尔伯特与阿克曼》中用斜体恰如其分地做了表述："判定性问题应该作为数理逻辑中的主要问题来看待。"（*Entscheidungsproblem muß als das Hauptproblem der mathematischen Logik bezeichnet werden.*）[36]哥德尔的不完备性定理并没有指出判定过程是不存在的，它只是证明了这样的判定过程不能确定任意公式是否为真。充其量，它只能确定一个公式的可证明性。

《希尔伯特与阿克曼》这本书用了9页篇幅讨论一阶谓词逻辑中的判定性问题，并且其中一半的篇幅讨论的是关于"特定情况下的判定性问题的解"。对于那些标准的（并且常见的）数理逻辑的公式来讲，判定过程已经开发出来了。所以看起来，通用的判定过程也应该是存在的。

但事实并非如此。1936年，美国数学家阿隆佐·邱奇（1903—1995）总结道（原文同样刻意使用了斜体）："一般情况下的一阶谓词逻辑的判定性问题是不可解的。"（*The general case of the Entscheidungsproblem of the engere Funktionenkalkül* [first-order predicate logic] *is unsolvable*.）[37]

阿兰·图灵独立于邱奇，应用了完全不同的方法，得到了同样的结论："希尔伯特的判定性问题是无解的。"[38]这是他在论文开始就提出来的，最终也作为了论文的结论："因此判定性问题不可解。"[39]

在邱奇和图灵发表他们的研究成果时，希尔伯特已经74岁了。那时，就连希尔伯特自己也被纳粹怀疑，仅仅因为他的名字是大卫。[40]希尔伯特生命中的最后一年在孤独和老迈中度过。他于1943年去世。在哥廷根大学希尔伯特的墓碑上刻着这样的文字：

Wir müssen wissen.
Wir werden wissen.

我们必须知道。我们必将知道。只是现在当人们读到希尔伯特的这段话时，他们能想到的仅仅是哥德尔、邱奇和图灵，以及不完备性和不可判定性。

希尔伯特的家乡哥尼斯堡在战争中被英国人的炸弹严重毁坏了。1945年并入前苏联，改名为加里宁格勒。由于坐落在波罗的海的海边，这座城市成为一个理想的海军基地，因此几十年来未曾对外开放。前苏

联解体之后，加里宁格勒归属俄罗斯，但是它被其他国家隔开，夹在了立陶宛和波兰之间，犯罪率出了名的高。

千年虫问题——有些人预测它会为20世纪带来终极的大灾难，并没有产生多大的影响。《纽约时报》在2000年第一个早晨的报纸头版上写着：

1/1/00：
技术和2000

巨大的安慰
计算机在’00年首个小时获胜

计算机程序员并没有真的在大量的关键系统中埋入时间炸弹。程序员们通常聪明得多！此外，他们还花费了很长时间努力搜索并解决了许多隐藏的问题。实际上，修改计算机程序通常来讲是很简单的，这就是为什么称之为软件。

计算机程序是以文本文件的形式编写和维护的，它们称为源代码。这些文本文件本身能够被其他程序读取和解析。程序员还可以编写特殊的程序，用来检查已有的源代码并找出可能存在问题的地方。比如，这样的程序可以搜索包含字符“year”或者“yr”的变量名，然后，程序员可以人工进行检查，看看相关程序是如何处理日历年的。

随着这些潜在的千年虫故障被找到并消除，一定有人开始思考一个更有野心的计划：能不能写出一个程序用来分析其他程序并且找到其他程序的问题？当然，这样的程序会非常复杂，编写起来非常困难，然而一旦能写出，就可以用来修正任意其他的程序，而这是非常有价值的。

是的，这会很困难，但是它在理论上可行吗？

答案是否定的。开发一个通用的故障发现算法是不可能的，这也是阿兰·图灵关于可计算数和判定性问题的论文所带来的一个令人不安的推断。

---

[1] 《连线》第7卷第4期（1999年4月），存档于www.wired.com/wired/archive/7.04。

[2] 托马斯·海尔斯爵士，*The Thirteen Books of Euclid’s Elements*，2E（Cambridge University Press，1926，Dover Publications，1956），Vol. 1，154-155。

[3] 托马斯·海尔斯爵士，*Mathematics in Aristotle*（Oxford University Press，1949，Thoemmes Press，1998），50-57，或者参见Howard Eves，*Foundations and Fundamental Concepts of Mathematics*，3rd edition（PWS-Kent，1990；Dover，1997），29-32。

[4] 康斯坦斯·雷德，*Hilbert*（Springer-Verlag，1970，1996），57。
[5] 雷德，*Hilbert*，25。
[6] 英文版见Jean van Heijenoort，ed.，*From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*（Harvard University Press，1967），1-82。
[7] 雷德，*Hilbert*，69。
[8] 引自Ben H. Yandell，*The Honors Class: Hilbert's Problems and Their Solvers*（A. K. Peters，2002），389。演讲的英文版最初刊登在 *Bulletin of the American Mathematical Society*，Vol. 8（July 1902）， 由Mary Winston Newson翻译。修订版见Jeremy J. Gray，*The Hilbert Challenge*（Oxford University Press，2000），240。
[9] 引自Yandell，*The Honors Class*，395。
[10] 引自Gray，*The Hilbert Challenge*，58。
[11] 引自Yandell，*The Honors Class*，397。
[12] 引自Yandell，*The Honors Class*，406。在Ivor Grattan-Guinness的 "A Sideways Look at Hilbert's Twenty-three Problems of 1900"，*Notices of the American Mathematical Society*，Vol. 47，No. 7（August 2000），752-757中，翻译为 "丢番图方程可解性的判定性"。术语 "有理整数" 意为我们熟悉的普通整数。
[13] 见*Oxford English Dictionary*，2nd edition，I，313。 同见Donald E. Knuth所著 "*The Art of Computer Programming*"，Vol. 1，Fundamental Algorithms，3rd edition（Addison-Wesley，1997）的开篇。最有名的古老算法是欧几里得找出两个数的最大公约数的方法，但是术语 "Euclid's Algorithm" 直到20世纪初才首次使用。
[14] 来自德语中的空间和时间*Raum*和*Zeit*。——译者注
[15] 功利主义是伦理学中的一个理论。——译者注
[16] 伯特兰·罗素，*The Autobiography of Bertrand Russell，1872-1914*（George Allen and Unwin Ltd，1967），36。
[17] *From Frege to Gödel*，124-125。
[18] 伯特兰·罗素，"The Theory of Types" 于 *From Frege to Gödel*，150-182中。
[19] I. Grattan-Guinness，*The Search for Mathematical Roots, 1870-1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel*（Princeton University Press，2000），380。
[20] Grattan-Guinness，*The Search for Mathematical Roots*，454。
[21] 最初的结论出现在*54.43节，"From this proposition it will follow, when arithmetical addition has been defined, that 1 + 1 = 2"，另外可以在怀

特海和罗素所著的 *Principia Mathematica to*56*（Cambridge University Press，1997）简本中方便找到。不过，通过严谨的观察完成的证明直到 Vol. II *110.643节中才出现，“上述引理有时是有用的”。

[22] 雷德，*Hilbert*，137。

[23] 后来数学家G. H. 哈代写了一本描述这些事件的小册子，1942年由 Cambridge University Press出版，并在小圈子里传播。后来重新出版，书名改为 *Bertrand Russell and Trinity: A College Controversy of the Last War*（Cambridge University Press，1970）。

[24] Grattan-Guinness，*Search for Mathematical Roots*，471，footnote 28。

[25] William Ewald，ed.，*From Kant to Hilbert: A Source Book in the Foundations of Mathematics*（Oxford University Press，1996），Vol. II，1113。

[26] 引自 Paolo Mancosu，“Between Russell and Hilbert: Behmann on the Foundations of Mathematics”，*The Bulletin of Symbolic Logic*，Vol. 5，No. 3（Sept.1999），321。

[27] Wilfried Sieg，“Hilbert’s Programs 1917-1922”，*The Bulletin of Symbolic Logic*，Vol. 5，No. 1（March 1999），22。

[28] 《希尔伯特和阿克曼》，*Grundzüge der Theoretischen Logik*（Verlag von Julius Springer，1928）。第2版于1938年发表，其中反映了前几十年中的其他研究，德文第2版的英文翻译由Chelsea 出版公司于1950年出版。德文第1版没有英文翻译版。

[29] 《希尔伯特与阿克曼》，*Grundzüge der Theoretischen Logik*，68。

[30] *From Kant to Hilbert*，Vol. II，1157-1165。

[31] 实际标题是 “Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I”，发表在 *Monatshefte für Mathematik Und Physik*，Vol. 38（1931），173-198。 第一个英文版是由爱丁堡大学的Bernard Meltzer翻译，出现在书Kurt Gödel，*On Formally Undecidable Propositions of Principia Mathematica and Related Systems*（Basic Books，1962，Dover Publications，1992）中。第二个英文版由纽约皇后大学的Elliott Mendelson教授翻译，出现在Martin Davis 的书 *The Undecidable: Basic Papers on Undecidable Propositions，Unsolvable Problems and Computable Functions*（Raven Press，1965），5-38中。第三个英文版由Jean van Heijenoort翻译（Gödel本人也参与协助），出现在他自己的书 *From Frege to Gödel*中，596-616。这也是Kurt Gödel，*Collected Works*，*Volume I*，*1929-1936*（Oxford University Press，1986），144-195中所使用的翻译版本。这篇论文常称为 “Gödel

1931”。
[32] 雷德，*Hilbert*，198。
[33] *The Autobiography of Bertrand Russell*，1872-1914，153。
[34] http://nobelprize.org/nobel_prizes/literature/laureates/1950。
[35] 雷德，*Hilbert*，205。
[36] 《希尔伯特与阿克曼》，*Grundzüge der Theoretischen Logik*，77。
[37] 阿隆佐·邱奇，“A Note on the Entscheidungsproblem”，*The Journal of Symbolic Logic*，Vol. 1，No.1（March 1936），41。
[38] 阿兰·图灵，“On Computable Numbers, with an Application to the Entscheidungsproblem”，*Proceedings of the London Mathematical Society*，2nd Series，Vol. 42（1936），231。（本书第67页）。
[39] 同上，262（本书第277页）。
[40] 雷德，*Hilbert*，209。

第二部分
可计算数

第4章 图灵的学业

阿兰·图灵10岁时得到一本埃德温·坦尼·布鲁斯特所著的《每个儿童应该知道的自然奇观》。图灵后来说[1]，这本书开启了他的科学视野，并对他理解人与机器之间的关系产生了更深刻的影响。“显然，人体也是一台机器。”那本书对此解释道：

> 它是一台极其复杂的机器。虽然比任何手工制作的机器都要复杂千万倍，但其本质上仍然是一台机器。有人曾将人体比作一台蒸汽机，但那时我们还不太了解它的工作原理。现在，我们会把它比喻为一台内燃机，就像是汽车、轮船和飞机的内燃机一样。[2]

20世纪初，“人体是机器”的想法被看成是非常无知的，就像现在儿童读物里很幼稚的想法一样。但事实并非如此。在阿兰·图灵出生前200年，法国医生兼哲学家朱利安·奥佛雷·拉·美特利（1709—1751）在其1747年的争议性作品*L’Homme Machine*（《机械人》）[3]中，毫不掩饰地描述了人体甚至思维的机械般的工作机制。阿兰·图灵从小就觉得自己的身体也是一台机器，后来也因探索机器和人类间的联系而被世人铭记。

阿兰·麦席森·图灵于1912年6月23日出生在伦敦帕丁顿的疗养院里。他的父亲曾在印度公务署为英帝国效力，母亲出生在马德拉斯，外祖父是一位在印度修建桥梁和铁路而赚了大钱的工程师。1907年，图灵的父母在一艘从印度到英国的船上相遇，同年他们在都柏林结婚。1908年年初，他们回到印度。阿兰是他们的第二个男孩，他母亲1911年在印度怀上了他，但他出生在英国。

在幼年，阿兰和他的哥哥约翰被留在英国，由一对退休夫妇照顾，而他们的父母住在印度，这在当时很常见。1922年，阿兰进入肯特的哈兹勒赫斯特预备学校学习。他最初的兴趣是地图、国际象棋和化学。[4] 1926年，他被一所最古老的英国公立学校舍伯恩录取。图灵在舍伯恩第一学期的第一天被大罢工所阻，不能乘火车去学校。于是，阿兰决

定骑车60英里上学，这一壮举被当地的报纸所报道。[5]

在舍伯恩，阿兰没能与其他男孩打成一片。他害羞、孤独，似乎总是衣衫不整、墨迹斑斑。“他的所有特征都容易成为笑柄，尤其是他那害羞、犹豫、尖细的声音——不完全是口吃，而是吞吞吐吐，就像在等待一个复杂的程序将他的想法转化成人类语言一样。”[6]他本可以在学习上表现优异而弥补自己的不足，但事实并非如此。只有在数学上，他才表现出一些智力天赋的端倪。

到了1929年，阿兰开始着迷于《物理世界的自然》（1928）一书。这是一本广为流行并极具影响力的书，由剑桥大学天文学家亚瑟·埃丁顿爵士所著，书中探讨了相对论和量子理论的新科学所带来的影响。阿兰同时和一个名为克里斯托弗·莫科姆的同学交往密切，他和阿兰在科学和数学上有着共同的兴趣，而且出生在一个比阿兰家更有意思并兼具科学气氛的家庭。克里斯托弗的外祖父是约瑟夫·斯万爵士，他在1879年发明了白炽灯泡，独立于爱迪生的发明。

回想起来，阿兰·图灵很可能在那时发现了他的同性恋倾向，克里斯托弗是他的初恋。但是没有任何迹象表明，这两名青年之间发生了身体接触，他们一起做化学实验，交流数学公式，并探讨埃丁顿和剑桥大学另一位天文学教授詹姆斯·简爵士所著书中的新天文学和新物理学。

剑桥大学是有抱负的英国科学家追逐向往之地，其在科学和数学上最享有盛名的学院就是三一学院。1929年12月，阿兰和克里斯托弗花了一周的时间到剑桥大学参加奖学金考试，一起沐浴在弗朗西斯·培根、艾萨克·牛顿、詹姆斯·克拉克·麦克斯韦母校的氛围中。他们回到舍伯恩一周后，考试结果公布在了《泰晤士报》上。阿兰没被录取，而克里斯托弗被录取了。克里斯托弗将前往三一学院，而阿兰最大的希望是能争取在下一年入学三一学院或者剑桥的其他学院。

两个月后，克里斯托弗突然生病并在一周内去世，病因是他小时候所感染的牛结核病。他们舍伯恩的一位旧日同窗在信中写道：“可怜的图灵因为这个打击几乎崩溃，他们一定是极其要好的朋友。”[7]虽然阿兰·图灵也与其他男人有着更亲密的性关系，但显然他对克里斯托弗的爱与崇拜是其他人所不能比的。

1930年12月，图灵再次参加了三一学院的考试，但仍然未被录取。他的第二选择是剑桥大学国王学院。这一次，他决定专攻数学，全心钻研G. H. 哈代的经典著作《纯数学教程》[8]（*A Course of Pure Mathematics*）备考，这本书在当时已经是第15版了。1931年秋，阿兰开始了他在剑桥大学国王学院的学习。

接下来的一年，图灵研究起一本叫做《量子力学的数学基础》（*Mathematische Grundlagen der Quantenmechanik*）的新书，这本书由

年轻的匈牙利数学家约翰·冯·诺依曼所著。20世纪20年代中期，冯·诺依曼曾与大卫·希尔伯特在哥廷根大学一起共事。绝大多数早期量子力学的数学研究工作都是在哥根廷大学进行的。20世纪30年代，冯·诺依曼移民美国并在普林斯顿大学任教，1933年成为普林斯顿高等研究院聘任的首批数学家之一。现在，在几个有趣的地方，约翰·冯·诺依曼和阿兰·图灵的生活有了交集。

图灵与冯·诺依曼的第一次见面很可能是在1935年夏天，当时冯·诺依曼利用在普林斯顿大学的工作假期来到剑桥大学做关于殆周期函数的演讲。图灵已经熟知演讲的主题以及冯·诺依曼在这方面的研究工作。就在那年春天，图灵已经发表了他的第一篇论文，共两页，讨论了”左右殆周期性的等价性”（Equivalence of Left and Right Almost Periodicity，伦敦数学学会，1935），推广了冯·诺依曼在前一年发表的一篇论文。

他们都没想到，两人会在次年于新泽西州的普林斯顿再次相遇。

图灵对于数理逻辑这一精妙深奥领域的兴趣可能开始于1933年，当时他阅读了伯特兰·罗素1919年的作品《数学哲学导论》。书的末尾写道：

> 如果有学生因为这本书而迈入数理逻辑的大门，并进行认真的研究，那么这本书就达到当时写作的初衷了。[9]

1935年的春季学期，图灵修读了”数学基础”课程，授课人是麦克斯韦·赫尔曼·亚历山大·纽曼（1897—1984），其姓名缩写M. H. A.。纽曼更为人熟知，人们常亲切地称他麦克斯。麦克斯·纽曼名声在外的是他在组合拓扑方面的工作，不过他也可能是剑桥大学在数理逻辑方面最有见识的人。纽曼整个课程的高潮是对哥德尔不完备性定理的证明。（研究生水平的数理逻辑导论课程至今仍然采用类似的结构。）

此外，纽曼的课程也涵盖了尚未解决的判定性问题。”是否有一种确定的方法，或者纽曼所说的’机械过程’，它可以应用于一个数学命题，并得出该命题能否被证明的结论？”[10]当然，对于”机械过程”，纽曼指的不是一台机器。机器也许能够进行简单的算术，但几乎不能解决实际意义上的数学问题。纽曼暗指的是后人称为”算法”的一类过程——用于解决某个问题的一组明确（但无意识的、非智能的）指令集。图灵开始研究判定性问题很可能是在1935年初夏。[11] 那时，他已经获得了剑桥大学奖学金，每年300英镑。图灵后来说，想到判定性问题的解决思路时，他正躺在格兰切斯特草坪上，这是剑桥学生很喜欢的一个休闲场所，距国王学院大约两英里。

到1936年4月，图灵把论文”论可计算数及其在判定性问题上的应用”[12]的草稿交给了纽曼。

图灵的论文采用了一种不同寻常的数学证明方法。一开始，他描述了一个虚构的可以做一些简单操作的计算机器。尽管这个机器很简单，但是图灵断言它在功能上等价于一个进行数学运算的人。他设置好这些机器来计算数字。作为示例，图灵给出的第一台机器可以计算1/3的二进制形式（.010101…），第二台机器可以计算一个无理并很可能也是超越数的数（.001011011101111…）。他说服我们，还可以定义出机器来计算π、e及其他有名的数学常量。图灵甚至创造了一个通用机器，它能模拟其他任何一台计算机器的操作。

然而，图灵机——这些假想装备的后来叫法——无法计算每个实数。他设计的机器只能进行有限数量的操作，通过用数字表示这些操作，他指出每一台机器唯一地用一个整数来描述，我们把这个整数称为描述数（Description Number）。因此，图灵机是可数的，可计算数——图灵机可以计算的数——也一定是可数的，但实数是不可数的（从康托尔的证明中可知）。可计算数当然包括代数数，而且还包括π和e等超越数，但由于可计算数是可数的，因而它们根本无法涵盖所有的实数。

图灵机也是会出错的，我们完全可以定义一个根本不会正确工作或者不会做任何有意义工作的图灵机。图灵将他定义的机器分为”符合要求的”和”不符合要求的”两种。

由于图灵机完全由描述数定义，我们也许有可能创造一台这样的图灵机，它能够分析这些描述数，以确定某一特定的机器是否是符合要求的。图灵证明了这是不可能的：没有一种判定图灵机是否符合要求的通用过程。一台图灵机可以分析另一台图灵机的唯一方式，是一步一步地跟踪机器的操作。总之，你必须实际运行一台机器，以确定它接下来会干什么。

图灵机的这些性质也适用于计算机程序。一般情况下，一个计算机程序不可能分析另一个计算机程序，除非一步一步地模拟它的运行。

图灵还证明了，我们甚至无法定义图灵机去做一些看似简单的事，例如确定另一台机器是否会打印数字0。在其论文的最后一节（将在本书第三部分讨论），图灵构造了一个数理逻辑上的命题，它等价于判定一个特定的图灵机是否将会打印数字0。由于他已经得出这样的”判定”是不可能的，所以这个命题在逻辑上是不可证明的，因此”判定性问题不可解”（图灵论文第262页，本书263页）。

大约在麦克斯·纽曼阅读图灵论文手稿的同一时间，他又收到美国数学家阿隆索·邱奇寄来的短论文”判定性问题的笔记”[13]的单行本。基

于已刊出的另一篇论文[14]，邱奇的文章同样做出了判定性问题不可解的结论。

别人比图灵捷足先登了。这通常意味着他的论文不能发表，注定要被遗忘。但麦克斯·纽曼意识到，图灵的方法更具创新性，并且与邱奇的方法有着很大的差异。他仍然建议图灵向伦敦数学学会提交论文发表。（从发表的论文看，该学会于1936年5月28日收到它。）图灵在5月29日给他母亲的信上对此做出了解释：

> 现在，有一篇论文同时在美国发表，作者是阿隆索·邱奇，他和我做的事相同，只是方法不同。尽管如此，纽曼先生和我觉得，截然不同的方法完全能够让我的论文得以发表。阿隆索·邱奇住在普林斯顿，所以我已经相当确定，我将去那里。[15]

5月31日，麦克斯·纽曼同时写信给阿隆索·邱奇和伦敦数学学会的秘书长。在给邱奇的信中，他写道：

> 在你送给我的论文单行本中，你定义了”可计算数”（Calculable Numbers），并指出了希尔伯特逻辑上的判定性问题是无法解决的，这一问题也是另一位年轻人所苦苦思索的。这个年轻人就是阿兰·图灵，他正想发表一篇论文，其中出于同样的目的定义了”可计算数”（Computable Numbers）。在他的处理方法中，涉及一种能产生任何一个可计算序列的机器，这与你的方法很不同，但看上去优点很多。我觉得如果可能，明年他应该和你在一起研究。[16]

在给伦敦数学学会秘书长F. P. 怀特的信中，纽曼写道：

> 我想你已经知道了图灵关于可计算数的论文。正当这篇论文完成并准备发表时，我收到了来自普林斯顿的阿隆索·邱奇的单行本，这篇文章在很大程度上率先给出了图灵论文的结果。
>
> 尽管如此，我仍然希望你们能发表图灵的论文，因为他们的方法极其不同。而且，由于这个结果非常之重要，因而我们应该关注不同的处理方法。邱奇和图灵的论文的主要结果就是希尔伯特的追随者们研究了多年的判定性问题，即找到一种机械的方法以判定一行给定的符号所表述的是否是一条可由希尔伯特公理证明的定理，它的一般形式是不可解的。[17]

图灵增加了一个附录，主要阐述他的计算性（Computability）理念和邱奇的“有效计算性”是等价的。伦敦数学学会于1936年8月28日收到

这个附录。

图灵的论文发表在伦敦数学学会1936年11月和12月的论文集里[18]，1937年12月发表了[19]一份三页纸的修订稿。阿隆索·邱奇在1937年5月的《符号逻辑杂志》（*Journal of Symbolic Logic*）中针对这篇论文写了一篇只有四段的评论，其中写道：“一位持有铅笔、纸和一串明确指令的人类计算者，可以被看做是一种图灵机。”[20]这是已知的“图灵机”一词最早见诸文字的地方。

图灵的论文分为11章及一个附录。论文开头的引言直接开始描述图灵构想的这一类新的数。

[230]

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936. -- Read 12 November, 1936.]

The “computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means.

论可计算数及其在判定性问题上的应用

A. M. Turing

[1936年5月28日收到。1936年11月12日审完。]

可计算数可以被简单描述为其小数表达式可在有限步骤内计算出来的实数。

图灵只考虑实数集，他指出，可计算数是实数集的一个子集，这也意味着存在一些不可计算的实数。当然，这一点并不显然。

对于“小数表达式”，图灵意指1/3应该表示为0.33333…，π应该计算成3.14159…，这乍看起来与他的“有限步骤”这一理念相冲突。显然，我们永远不能真正算完1/3或π的小数。然而，在图灵的论文中，“步骤”并不是指确定数位的实际过程，而是指确定数位的方法。用诸如“下一位是4，下一位是7，下一位是0……”的方法，显然可以计算任何实数，但这不是一个有限的方法。1/3和π都可以用一套算法计算出来（其中一个的算法更简单一些），我们计算它们所采用的步骤（长除法或更复杂的方法）只用到有限数量的规则。

Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computabl functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers.

尽管从表面上看本论文的主题是可计算数，但是我们很容易用几乎相同的方法定义和研究关于整数变量，或者实数、可计算数变量的可计算函数，可计算谓词等，每种情况下涉及的基本问题都一样。我选择可计算数进行具体的研究，是因为它所涉及的技术细节最简单。我希望可以很快就给出可计算数、可计算函数等之间的关系，这将包括一套用可计算数表示实变函数的理论。

图灵并未按其论文说的继续写下去。哥德尔也打算续写自己关于不完备性的著名论文，甚至预先在标题里加了罗马数字Ⅰ，但他从未写过续篇，因为他的论文很快就被接受了，超出了他的预想。图灵则是因为后来开始关注其他事情了。

图灵用下面这句陈述总结了论文的第一段。

According to my definition, a number is computable if its decimal can be written down by a machine.

根据我的定义，如果一个数的小数形式可以被机器写下来，那么它就是可计算的。

在1936年，这样的说法非常奇怪，因为当时并不存在一般意义上图灵所要求的那种机器。

图灵很可能知道英国数学家查尔斯·巴贝奇（1791—1871）的一些

研究。巴贝奇曾设计了一个“差分机”（Difference Engine）用来计算对数表，然后大约在1833年放弃了这个项目，转而研究一种更像通用计算机的“分析机”（Analytical Engine）。巴贝奇也曾在剑桥工作，其未完成的机器上的零件在肯辛顿的科学博物馆展出过。不过，图灵似乎一点都未受到巴贝奇的概念或术语的影响。

图灵可能知道微分分析仪（Differential Analyzer），也可能不知道。自1927年起，万尼瓦尔·布什（1890—1974）和他在MIT的学生开始研制这台机器。但是，这个模拟的（非数字式的）计算机主要用来解决工程应用中的微分方程。从数学或工程角度来讲，图灵或许会对这台机器感兴趣，不过这台机器并不能为这一特定问题提供很大帮助。

很难想象在20世纪30年代中期，图灵如何能知道其他的早期计算机项目。图灵肯定不知道工程专业学生康拉德·楚泽（1910—1995）自1935年起开始在其父母位于柏林的公寓里建造计算机。乔治·斯蒂比兹（1904—1995）曾利用他从贝尔实验室拿回家的一些电话继电器搭接二进制加法器，不过这已经是1937年图灵论文发表之后的事情了。也是在1937年，哈佛大学研究生霍华德·艾肯（1900—1973）开始探索自动计算，从而促使哈佛大学与IBM合作，创造了哈佛马克一型计算机（Harvard Mark I）。[21]

这个时期，在解决可计算性问题和希尔伯特的判定性问题的先驱中，图灵是其中之一，其余人包括阿隆索·邱奇、埃米尔·波斯特（1897—1954）、斯蒂芬·克莱尼（1909—1994）。[22]在20世纪30年代中期研究自动计算的那些人里，图灵也算其一。

图灵总结了出现在论文后面章节中的部分结论，他写道：

> In §§9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers *π*, *e*, etc.

> 在§9、§10中，我会给出一些论证，说明可计算数也包括那些理当被认为是可计算的数。特别地，我会指出有几大类的数都是可计算的，例如所有代数数的实部、贝塞尔函数零点的实部、数π和e，等等。

“理当被认为是可计算的数”就是人们实际生活中计算的，并且存在相应算法的数。图灵根本不屑提及所有的有理数都是可计算的，因为这是显而易见的。他很快便把代数数也加进了可计算数的行列（他认为只有代数数的实部符合要求，因为代数方程的解有可能有实部和虚部两部分，而他已经限制了可计算数为实数）。

图灵断言了代数数是可计算数，随后开始在超越数领域讨论这一问题。不过，他说只有某些超越数是可计算的。贝塞尔函数是特定形式的微分方程的解。零点就是函数取值为0的点。它们曾被刊印成表，因此被认为是可计算的。（现在，需要用这些数时，一般可以用计算机程序来计算。）虽然图灵没有提到，但三角函数和对数函数的一般取值都是超越数，这些数也是可计算数。同样，π、e等常数也如此。

图灵并没有声称所有的超越数都是可计算的，否则可计算数就会和实数一样了。

> The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

> 尽管如此，可计算数并不完全包括所有可定义的数，我们将给出一个可定义的却不可计算的数。

我们就先在这儿留下一个悬念吧。到时候图灵将定义一个他（以及他的机器）不能计算的数。

现在，图灵谈及了实数与可计算数之间差异的关键。

> Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable.

> 尽管可计算数如此之多，并且在很多方面与实数相似，但它是可数的。

可计算数是可数的。可计算数的可数性显示了它们与实数的不同，因为实数是不可数的。

In §8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel[†].

[†] Gödel, “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I” *Monatshefte Math. Phys.*, 38 (1931), 173-198.

在§8中，我将仔细考察几个论证，它们似乎能证明出与此相反的结论。正确地应用其中的某个论证，可以得到与哥德尔的结论[†]大致相似的结论。

[†] 参见Gödel，“Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I”，*Monatshefte Math. Phys.*，38（1931），173-198。

这就是著名的哥德尔不完备性定理。注意，图灵的脚注引用的是哥德尔论文的德文标题，英文译本直到1962年才出版。

These results

[231]

have valuable applications. In particular, it is shown (§11) that the Hilbertian Entscheidungsproblem can have no solution.

这些结果的应用价值很大。特别地，它表明希尔伯特的判定性问题是无解的（§11）。

这是接下来的18页论文中最后一次提到希尔伯特。

图灵在了解了阿隆索·邱奇的证明，并断定两种方法是等价的之后，觉得需要为他的论文增加一个附录。引言的最后一段也是在那时加进来的。

In a recent paper Alonzo Church[†] has introduced an idea of “effective calculability", which is equivalent to my “computability”, but

is very differently defined. Church also reaches similar conclusions about the Entscheidungsrproblem[‡]. The proof of equivalence between “computability” and “effective calculability” is outlined in an appendix to the present paper.

[†] Alonzo Church, “An unsolvable problem of elementary number theory”, *American J. of Math.*, 58 (1936), 345-363.

[‡] Alonzo Church, “A note on the Entscheidungsproblem”, *J. of Symbolic Logic*, 1 (1936), 40-41.

阿隆索·邱奇在其近期的论文中[†]引入了“有效可计算性”（effective calculability）的概念，这和我的“可计算性”（computability）是等价的，但定义方式十分不同。邱奇也就判定性问题得出了相同的结论[‡]。关于“可计算性“和”有效可计算性“的等价性证明将在本文的附录中给出。

[†] 参见Alonzo Church，“An unsolvable problem of elementary number theory”，*American J. of Math.*，58（1936），345-363。

[‡] 参见Alonzo Church，“A note on the Entscheidungsproblem”，*J. of Symbolic Logic*，1（1936），40-41。

这是图灵的论文，接下来的28页中最后一次提到判定性问题。《牛津英语字典（第2版）》指出，除了1889年的一本字典，上述文字首次使用了“可计算性”一词。自那之后，30多本书的书名中使用了“可计算性”，第一本书是马丁·戴维斯的《可计算性和不可解性》，由McGraw-Hill出版社于1958年出版。

图灵的论文有11节，第1节现在开始。

1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach §9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

1.计算机器

我们已经提到，可计算数是那些小数表达式可以通过有限步骤计算出来的数。这一概念需要更加明确的定义。在§9之前，我们不会真正尝试证明这个定义的合理性。就目前而言，我只能说，合理性的证明依赖于人类记忆的有限性。

图灵曾说可计算数就是那些可以被机器写下来的数，而现在又用人类记忆的有限性来解释定义中的“有限步骤”。将人与机器随意地关联起来，这种做法是图灵研究的一大特点。

图灵最初说，可计算数是可以通过有限步骤计算出来的，当时听着确实很有道理。但是，现在他要通过人类思维的有限性来解读它，这就提出了关于数学真实性的本质问题。我们称实数为“实”数，尽管事实上绝大多数的数从没有人见过。此外，图灵还将在论文中指出，大部分实数都不能通过有限的算法计算出来。实数究竟在什么意义上存在呢？这是个哲学问题，对此，图灵也只是在其论文的修订版中模糊地提及（见本书第16章）。

人类思维的状态是离散的。据此，图灵将人与机器关联了起来。

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q_1, q_2, …, q_R which will be called “*m*-configurations”.

我们可以将一个正在进行实数计算的人比作一台只能处理有限种情况q_1, q_2, q_3,…, q_R的机器，这些情况称为“m-格局”。

这里，m代表机器（machine），一台机器有有限个格局，并根据当前格局做不同的事情。更现代的术语则是状态（state），后来图灵引用“思维状态”来类比这些机器状态。举个例子，一台简单的洗衣机有注水、洗涤、漂洗、甩干四个状态。作长除法同样也涉及一系列不同的大脑格局或者思维状态：“现在要作乘法”、“现在要作减法”、“现在要借位”机器的运转实际上就是在不同的格局间来回切换，通常以一种重复的方式进行。

The machine is supplied with a “tape” (the analogue of paper)

running through it, and divided into sections (called “squares”) each capable of bearing a “symbol”.

该机器配有“纸带”（可以与纸类比）。纸带穿过机器运转，同时被分成一个个区段（称作“方格”），每个方格中都可以放置一个符号。

图灵说这种纸带“可以与纸类比”，因为人就是在纸上计算数字的。图灵机中的纸带往往被想象成真的纸带，但如果真要构建一台图灵机，那么这张纸带或许会是上了磁的，或者仅仅是计算机内存中的一块。

人类通常使用二维的纸张，而图灵限制他的机器只能用分成一格一格的一维纸带。方格中的字符可以是十进制的0到9，或者可以包括字母表中所有的英文字母，抑或键盘上的所有95个字符。（正如你将看到的，图灵甚至允许一个“符号”包含多个字符。）

为了在这一节论文中表示这些符号，图灵使用了大写的德文哥特式字体S（意指“symbol”），它看起来是这样的：$\mathfrak{S}$。你将不止一次看到这个符号。

At any moment there is just one square, say the *r*-th, bearing the symbol $\mathfrak{S}(r)$ which is “in the machine”.

任何时刻，只有一个方格，比如说第*r*个，它里边的符号$\mathfrak{S}(r)$是“在机器里”的。

这里，图灵假设纸带的方格可以通过编号来识别。例如$\mathfrak{S}(3451)$代表纸带第3451个方格上的符号。如果那个方格中包含字符“A”，那么

 (3451) 就是“A”。但是严格地说，纸带的方格并没有被编号，机器也并不通过编号来找到特定的方格。（换言之，方格没有一个具体的地址。）

图灵说，在任何时候，纸带上只有一个方格“在机器里”，并可以被机器检测。

> We may call this square the “scanned square”. The symbol on the scanned square may be called the “scanned symbol”. The “scanned symbol” is the only one of which the machine is, so to speak, “directly aware”.

> 我们可以称这个方格为“扫描格”，扫描格中的符号可以称为“扫描符”。可以这么说，“扫描符”是机器当前唯一可以“直接感知”的符号。

机器不能一次“看到”整个纸带，它每次只能“看”纸带中的一个方格。

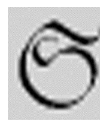
> However, by altering its *m*-configuration the machine can effectively remember some of the symbols which it has “seen” (scanned) previously.

> 尽管如此，通过调节m-格局，机器可以有效地记住之前“看到”（扫描）的字符。

一台机器可以从一种m-格局转换到另一种m-格局，这是由当前的扫描符决定的。例如，在特定的m-格局q_{34}下，如果扫描符是“A”，它可能转换到m-格局q_{17}；如果扫描符是“B”，它可能转换到m-格局q_{123}。因此，m-格局q_{17}就“知道”前一个扫描符为“A”，m-格局q_{123}就知道前一个扫描符为“B”。（这并不完全正确，其他格局也可能转换到q_{17}和q_{123}，但

是从机器的设计意图来看，q_{17}和q_{123}应该对之前的情况了解足够多，从而可以继续工作下去。）

The possible behaviour of the machine at any moment is determined by the *m*-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair q_n, $\mathfrak{S}(r)$ will be called the “configuration” thus the configuration determines the possible behaviour of the machine.

机器在任何时候可能的行为都是由当前的m-格局q_n和扫描符(r)决定的。这对q_n与$\mathfrak{S}(r)$的组合称为“格局”。因此，格局决定了机器可能的行为。

m-格局可以是q_1、q_2等，当m-格局与一个扫描符配对时，图灵简单地称之为格局（configuration）。

图灵已经暗示，机器会根据扫描符在m-格局之间的切换。这个机器还能做些其他什么呢？能做的并不多。

In some of the configurations in which the scanned square is blank (*i.e.* bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left.

在某些格局里，扫描格为空（即里边没有符号），机器会在这个扫描格写下一个新的符号，在其他格局中则会擦除这个扫描符。机器也可以改变正被扫描的方格，但只能移到左边或右边一格。

如果我把这个读写符号的机制叫做机器的读写头，我想我并没有违背图灵的意思。就好像录音机或录像机，读写头与纸带只在一个点相接触。图灵的读写头可以从纸带读取一个符号或者擦除一个符号，或者在其上写一个新的符号。它也可以向左或向右移动一格。（尽管读写头很可能是固定的，移动的是穿过机器的纸带，最好还是将读写头想象成是相对纸带移动的。）

> In addition to any of these operations the *m*-configuration may be changed. Some of the symbols written down
>
> [232]
>
> will form the sequence of figures which is the decimal of the real number which is being computed. The others are just rough notes to “assist the memory”. It will only be these rough notes which will be liable to erasure.

> 除了这些操作外，m-格局也可能会变化。有一部分写下的符号将组成一串数字序列，即被计算实数的小数表达；另一部分则是用来“协助记忆”的草稿。只有这些草稿才可以被擦除。

因为图灵希望他的机器能计算数，因此机器需要打印出数字，并且一般情况下，要打印一个无限长的数字序列。为了帮助这个过程本身顺利进行下去，机器可能需要把纸带的一部分作为某种便笺来使用。

图灵机是什么样子的呢？你可以想象一些模样古怪的机器[23]，不过更好的方法则是去照照镜子。正如一部著名科幻影片的高潮[24]所说，“图灵机就是人”——以一种非常受限却又十分精确的方式进行算法运算的人。

> It is my contention that these operations include all those which are used in the computation of a number.

> 我的观点是：这些操作包括了在计算数字中用到的所有操作。

也就是说，人在计算数字中用到的所有操作。如果你觉得这台机器

缺少一些基本的算术运算，比如加法或减法，那你说得一点没错。加法运算和减法运算没有内建在图灵机里。不过，只要有了正确的格局，图灵机便可以执行相应的算术运算。

The defence of this contention will be easier when the theory of the machines is familiar to the reader. In the next section I therefore proceed with the development of the theory and assume that it is understood what is meant by “machine”, “tape”, “scanned”, etc.

读者熟悉这一机器理论后，将会更容易理解我的这一观点。因此，下一节我将开始探讨这一理论，假设你已经知道了“机器”、“纸带”、“扫描”等词的含义。

大家或许已经准备好要看一些真正的机器了，但图灵还不想让我们如愿，他想先抛出一些定义。

2. *Definitions.*

Automatic machines.

If at each stage the motion of a machine (in the sense of §1) is *completely* determined by the configuration, we shall call the machine an “automatic machine” (or *a*-machine.)

For some purposes we might use machines (choice machines or *c*-machines) whose motion is only partially determined by the configuration (hence the use of the word “possible” in §1).

2. 定义

自动机

如果机器在每一阶段的动作（在§1的意义下）完全由格局所决定，我们称这样的机器为“自动机”（或*a*-机器）。

我们也可能出于某些目的，使用那些格局只能部分决定动作的机器（选择机或*c*-机器）。（因此，我们在§1里用了“可能”一词。）

当描述格局如何决定一台机器的行为时（本书第61页），图灵的表述是“机器可能的行为”。行为的描述可以是不完整的，因为在一些机器中，行为会因人的交互动作（机器之外的“操作者”）而改变。

> When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator. This would be the case if we were using machines to deal with axiomatic systems. In this paper I deal only with automatic machines, and will therefore often omit the prefix *a*-.

> 当这样的一台机器达到某种模糊的格局时，除非有机器之外操作者给出选择，否则它不会继续工作。我们用机器来处理公理系统时就会出现这种情况。在本文中，我只讨论自动机，因此会经常省略前缀*a*-。

图灵对自动机和选择机的划分，某种程度上使我们联想到划分程序的一个传统方法：将程序分为批处理的和交互式的。现在，我们进行交互计算的经历如此之多，以至于可能会忘记仍然有许多不需要等待用户的键盘输入和鼠标点击就能运行的计算机程序。

选择机虽然可能很有趣，但是它们在图灵的论文中无足轻重。图灵论文中自动机的行为完全由其自身的格局决定。

> *Computing machines.*
>
> If an *a*-machine prints two kinds of symbols, of which the first kind (called figures) consists entirely of 0 and 1 (the others being called symbols of the second kind), then the machine will be called a computing machine.

> 计算机器
>
> 如果一台自动机（*a*-机器）打印两种符号，第一类符号（称为数字）完全由0和1组成（其他符号称为第二类符号），那么这样的机器就称为计算机器。

在展示样机之前，图灵就已经把机器限制为只打印数字0和1，这也是用来表示二进制数[25]的两个数字。使用二进制数是明智之举，但这一点对于1937年的大多数读者来说远没有我们今天所想的那么显而易见。克劳德·E. 香农（1916—2001）在其1937年麻省理工学院的硕士论文《继电器与开关电路的符号分析》中描述了电路与布尔代数的等价性，他一定很推崇选择二进制。但在早期的计算机中使用二进制并不普遍，尽管康拉德·楚泽使用了二进制，但艾肯和斯蒂比兹的机器都是基于十进制的，ENIAC（1943—1945）也是一台十进制机器。直到香农1948年的一篇论文[26]里，bit（比特，binary digit的缩写）一词才首次出现。

图灵并不打算论证在其机器中使用二进制数的合理性，使用它的好处直到论文的第245页（本书第146页）才显现出来。但为了让大家尽快放下这些疑问，我将在下一章比较一些简单的二进制机器和十进制机器。

> If the machine is supplied with a blank tape and set in motion, starting from the correct initial *m*-configuration, the subsequence of the symbols printed by it which are of the first kind will be called the *sequence computed by the machine.*

> 如果给机器装上空白纸带，并且从正确的初始m-格局开始运转，那么机器打印出的第一类符号组成的子序列就叫做“机器计算出的序列”。

一台装有空白纸带的机器开始运作，打印出0和1（第一类符号）以及其他的符号（第二类符号）。这些0和1组成了计算出的序列。图灵将计算出的序列与计算出的数区分开来了。

> The real number whose expression as a binary decimal is obtained by prefacing this sequence by a decimal point is called the *number computed by the machine*.

> 在这个序列的最前面加上一个十进制小数点（decimal point），并把它当作一个二进制小数（binary decimal），所得的实

数就称为机器计算出的数。

某种程度上，这句话读起来有点费力，因为里面的术语并不完全正确。但是，我们需要原谅图灵用词的混淆，因为那时的人们根本不习惯讨论二进制数。即便在今天，那些熟悉二进制的人一般也不善于面对二进制小数。即使把Windows的计算器设为科学模式也没有用：把一个数转成二进制时，它会直接去掉小数部分。

“十进制”的英文“decimal”是从拉丁文的“ten”演变而来的，这个单词的使用仅限于十进制数。下面是十进制小数：

.25

.5

.75

十进制小数点（decimal point）把整数部分（如果有的话）和小数部分分开。

上面三个数的二进制表示为：

.01

.1

.11

那个点不是十进制的小数点，实际上应该称为二进制小数点（binary point）。

正如二进制整数部分的每一位上的数字代表2的幂，小数部分的每一位二进制数字代表2的负幂。

.1相当于2^{-1}的二进制形式或者十进制分数的1/2

.01相当于2^{-2}或者1/4

.001是2^{-3}或者1/8

.0001是2^{-4}或者1/16

.00001是2^{-5}或者1/32

等等，因此二进制数.10101就是：

$$1\times2^{-1}+0\times2^{-2}+1\times2^{-3}+0\times2^{-4}+1\times2^{-5}$$

这么写或许看着更清楚：

$$\frac{1}{2}+\frac{0}{4}+\frac{1}{8}+\frac{0}{16}+\frac{1}{32}$$

等价于十进制的21/32或者.65625。和十进制中一样，许多二进制的小数部分也是循环的。下面是1/3的二进制表示：

.01010101…

2/3是：

.10101010…

类似地：

1/5 是 .001100110011…

2/5 是 .011001100110…

3/5是 .100110011001…

4/5 是 .110011001100…

图灵的措词更加准确："在这个序列的最前面加上一个十进制小数点（decimal point），并把它当作一个二进制小数（binary decimal），所得的实数就称为机器计算出的数。"

趁现在还在解释这块，我们来进一步审视这句话："用机器计算出来的数是一个以二进制小数点开头的序列表达出来的二进制小数。"

例如，如果有一台图灵计算机器只打印出一个0和一个1，没有其他字符，那么"机器计算出的序列"就是

0 1

而"机器计算出的数"就是一个以二进制小数点开头的序列

.01

这就是1/4的二进制形式。

因为二进制小数点总是位于计算出的序列之前，所以图灵的机器将只会计算介于0与1之间的二进制数，不过这么小的范围已经足够用来研究可数性了。

> At any stage of the motion of the machine, the number of the scanned square, the complete sequence of all symbols on the tape, and the *m*-configuration will be said to describe the *complete configuration* at that stage.

> 在机器运转中的任何阶段，被扫描方格的标号、纸带上所有符号构成的序列以及m-格局，共同描述了这个阶段的完全格局。

这是图灵从不同角度讨论这些机器时出现的"格局"一词的第三种用法，将它们理清楚是很重要的：

〉 m-格局是机器的一种状态；

〉 格局由一个m-格局和一个扫描符组成；

〉 完全格局是整个纸带在某一时刻的"快照"，加上当前的m-格局及读写头的位置。

The changes of the machine and tape between successive complete configurations will be called the *moves* of the machine.

在相邻的两个完全格局之间机器和纸带发生的变化，称为机器的“移动”。

下面两个定义在论文的后面才用到。

[233]

Circular and circle-free machines.

If a computing machine never writes down more than a finite number of symbols of the first kind, it will be called *circular.* Otherwise it is said to be *circle-free.*

A machine will be circular if it reaches a configuration from which there is no possible move, or if it goes on moving, and possibly printing symbols of the second kind, but cannot print any more symbols of the first kind. The significance of the term “circular” will be explained in §8

循环机和非循环机

如果一台计算机器只能写下有限个第一类符号，它就被称为是循环的（circular），否则称为非循环的（circle-free）。

如果一台机器运行到了某个不能移动的格局上，或者它能继续移动并有可能打印出第二类符号但永远不能打印出第一类符号了，那么它就是循环的。我们将在§8中解释”循环”这个概念的重要意义。

我之前提到一台仅打印了0和1、不再打印其他字符的机器。它只打印了有限个数字，根据图灵的定义，它属于循环机。这台机器会在某处卡住并且不能再打印数字，这不是件好事。图灵希望这台机器能永不停息地打印数字。

非循环机是好的机器。一台只打印一个0和1、不再打印其他东西的机器不是非循环机。如果机器真的想要计算1/4的二进制形式，它就必须在打印完0和1后继续不停地打印0。

尽管图灵没有提到这个问题，但是他好像在暗示他的计算机器是从左至右打印数字的，正如我们从二进制小数点后读数字一样。

> *Computable sequences and numbers.*
> A sequence is said to be computable if it can be computed by a circle-free machine. A number is computable if it differs by an integer from the number computed by a circle-free machine.

> 可计算序列和可计算数
> 如果一个序列可以被非循环机计算出来，那么它就是可计算序列。如果一个数与非循环机计算出来的数只相差一个整数，那么它就是可计算数。

图灵在这里区分了序列和数。一个可计算的序列是：

010000…

对应的可计算数是：

.010000…

数

1.010000…

也可以认为是可计算的，因为它与机器计算出来的数只相差一个整数。数

10.01000…

也是如此。同样，负数也是可计算的。

> We shall avoid confusion by speaking more often of computable sequences than of computable numbers.

> 为了避免混淆，我们会更多地提及可计算序列，而非可计算数。

---

[1] 安德鲁·霍奇斯，*Alan Turing: The Enigma*（Simon & Schuster，

1983），11。所有图灵的传记信息来自这本书。
[2] 安德鲁·霍奇斯，*Alan Turing*，13。
[3] Julien Offray de La Mettrie，*Machine Man and Other Writings*，由Ann Thomson翻译并编辑（Cambridge University Press，1996）。
[4] 霍奇斯，*Alan Turing*，17。
[5] 霍奇斯，*Alan Turing*，21。
[6] 霍奇斯，*Alan Turing*，24。
[7] 霍奇斯，*Alan Turing*，46。
[8] 本书中文版已由人民邮电出版社于2009年出版。——编者注
[9] 罗素，*Introduction to Mathematical Philosophy*，second edition（George Allen & Unwin Ltd，1920；Dover Publications，1993），206。
[10] 霍奇斯，*Alan Turing*，93。
[11] 霍奇斯，*Alan Turing*，96。
[12] 霍奇斯，*Alan Turing*，109。
[13] 邱奇，"A Note on the Entschiedungsproblem"，*The Journal of Symbolic Logic*，Vol. 1，No. 1（Mar.1936），40-41。
[14] 邱奇，"An Unsolvable Problem of Elementary Number Theory"，*American Journal of Mathematics*，Vol. 58，No. 2（Apr.1936），345-363。邱奇的两篇文章都出现在Martin Davis，ed.，*The Undecidable*（Raven Press，1965）。
[15] 霍奇斯，*Alan Turing*，113。
[16] 霍奇斯，*Alan Turing*，112。
[17] 霍奇斯，*Alan Turing*，113。
[18] 这篇文章分在了回忆录的不同部分，前11页出现在Vol. 42第3部分（November 30，1936），剩下部分在Vol. 42第4部分（dated December 23，1936）。1937年，从1936年10月直到1937年4月陆续出版的部分汇集成第二系列，Vol. 42。这就是为什么图灵论文的出版日期中有的是1936年（各部分独立出版的日期）、1937年（Vol. 42完成的日期）或者1936~1937年（所有部分包含在Vol. 42中的日期）。
[19] 特别地，Vol. 43的第7部分（December 30，1937发行）出现在Vol. 43的第二系列中，Vol. 43也包括了1937年5月至12月出版的部分。
[20] 邱奇，"On Computable Numbers, with an Application to the Entscheidungsproblem"的审稿人，*The Journal of Symbolic Logic*，Vol. 2，No. 1（Mar.1937），42-43。
[21] 一本介绍这些早期计算机的优秀著作是Paul E. Ceruzzi著的*Reckoners*：*The Prehistory of the Digital Computer, from Relays to the*

Stored Program Concept，1935-1945（Greenwood Press，1983）。
[22] Robin Gandy，“The Confluence of Ideas in 1936”，in Rolf Herken，ed.，*The Universal Turing Machine: A Half-Century Survey*（Oxford University Press，1988），55-111；second edition（Springer-Verlag，1995），49-102。
[23] 其中最好的一个（配有各种配件的机器）出现在Gregory J. Chaitin的 “Computers, Paradoxes and the Foundations of Mathematics”，*American Scientist*，Vol. 90（March-April 2002），168。参见在线版http://www.cs.auckland.ac.nz/CDMTCS/chaitin/amsci.pdf。
[24] *Soylent Green*（1973）。
[25] 二进制数的概述可以查看本书作者的*Code*：*The Hidden Language of Computer Hardware and Software*（Microsoft Press，1999）。
[26] 香农，“A Mathematical Theory of Communication”*The Bell System Technical Journal*，Vol. 27（July，October 1948）。香农将这个新词归功于美国数学家J. W. Tukey，但是这个杜撰的词却得到了Lancelot Hogben 的恶评，详见 *The Vocabulary of Science*（Stein and Day，1970），146：“Tukey引入新词bit的做法，除了不负责任的谬赞之外什么也没得到。” 我不同意他的说法。比特在现代生活中如此普通，是个理想的词，就像那些事物本身一样。

第5章
运作的机器

毫无疑问，图灵意识到了，在一篇数学论文中介绍一台想象中的计算机器，这种做法既新颖又大胆。就像一位优秀的数学家所做的那样，他给出了这些机器的定义和形式化的描述。他并不需要给出什么例子，但我猜测，他也知道读者们必定不会仅仅满足于抽象的内容，他们需要实实在在的东西。他现在就来满足大家的这一期望。

3. *Examples of computing machines.*

I. A machine can be constructed to compute the sequence 010101….

3. 计算机器示例

Ⅰ. 可以构造一台计算序列010101…的机器。

这台机器打印出来的纸带会是这样：

…							0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	…

但也不完全是这样。就像图灵后面要解释的，他更希望他的机器只使用相间的方格来打印数字序列。第一台样机实际上打在纸带上的是这样：

…							0		1		0		1		0		1		0		1		0		…

图灵使用了德国哥特式字体的小写字母表示机器的m-格局。这些字母需要花点时间来适应，因此我会注意指出一些可能会带来麻烦的字母。图灵在他的第一台样机中使用的字母是b、c、k和e。（注意：德文的k看起来像f。）

The machine is to have the four *m*-configurations “$\mathfrak{b}$”, “$\mathfrak{c}$”, “$\mathfrak{f}$”, “$\mathfrak{e}$” and is capable of printing “0” and “1”. The behaviour of the machine is described in the following table in which “*R*” means “the machine moves so that it scans the square immediately on the right of the one it was scanning previously”. Similarly for “*L*”. “*E*” means “the scanned symbol is erased” and “*P*” stands for “prints”.

这台机器有四种m-格局：“$\mathfrak{b}$”，“$\mathfrak{c}$”，“$\mathfrak{f}$”，“$\mathfrak{e}$”并且可以打印0和1。下表描述了机器的行为，其中“*R*”指“移动机器使其扫描紧跟在刚才扫描的方格右边的那个方格”。类似的，“*L*”就指移动到左边一格，“*E*”表示“擦除扫描符”，“*P*”指“打印”。

在表中，*P*后面总是跟着要打印的字符。例如，*P*0指打印一个0，*P*1指打印一个1，而*Px*则表示打印一个*x*。

This table (and all succeeding tables of the same kind) is to be understood to mean that for a configuration described in the first two columns the operations in the third column are carried out successively, and the machine then goes over into the *m*-configuration described in the last column.

可以这样理解这个表（以及接下来所有这类表）：对于前两列中描述的格局，第三列的操作会被陆续执行。然后机器会转到最后一列的m-格局。

表有四列，分为两对

格局		行为	
m-格局	符号	操作	最终m-格局

机器的行为取决格局，也就是m-格局和被扫描方格中的符号。第三列包含了各项操作（只能是*P*、*E*、*L*和*R*）。第四列则是下一个m-格

局。

通常，第二列明确表示了一个特定的扫描符，如0或1，但是图灵也使用了“Any”这个词表示任何符号，也使用了“None”一词表示没有符号，也就是一个空白的方格。（对现在的程序员来说，这可能有点让人困惑，因为现在程序员已经习惯于将空格视为与其他字符一样的符号了。而当图灵使用“Any”时，他通常意指“任何非空格”的符号。）对于任何符号包括空格在内的情况，我们这样处理：

> When the second column is left blank, it is understood that the behaviour of the third and fourth columns applies for any symbol and for no symbol.

如果第二列留空，则理解为第三列和第四列的行为适用于任何符号以及没有符号的情形。

幸运的是，引发歧义的可能性非常小。

> The machine starts in the *m*-configuration with a blank tape.

机器装上一条空白纸带后，从m-格局开始运行。

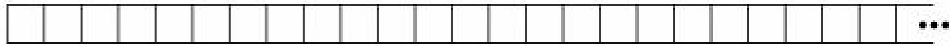
图灵的机器总是以m-格局（意为begin，或者更恰当地，）开始。下面便是我们期待已久的机器。

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$

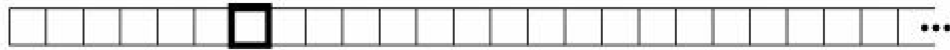
格局		行为	
m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$

可以像这样读这些行：对于m-格局 $\mathfrak{b}$ ，当扫描格为空（符号“None”）时，打印0，将读写头向右移动一格，然后转换到m-格局 $\mathfrak{c}$ 。

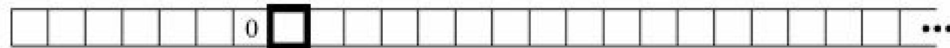
让我们启动这台机器，观察一下它是如何工作的。我们以m-格局 $\mathfrak{b}$ 和一条空白纸带开始。尽管理论上纸带两头都是无限延伸的，但是图灵这篇论文中描述的机器只需要纸带无限向右延伸即可，可计算序列中的数字就打在这些地方。



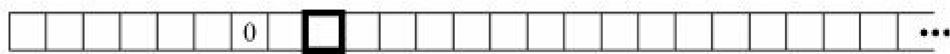
读写头可以用很多种记号来表示。我这里选择了一个围绕当前被扫描方格的粗边框。初始时可以把读写头放置在纸带的任意位置：



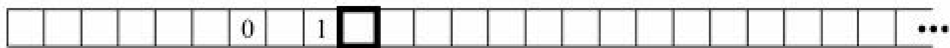
扫描格里面没有符号。上表告诉我们，对于m-格局 𝔟 ，如果没有符号，则机器打印0，然后右移一格。



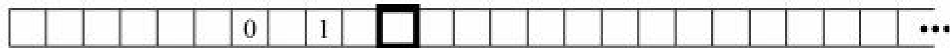
新的m-格局是 𝔠 。如果方格为空，右移一格，转到m-格局 𝔢 ：



对于m-格局 𝔢 ，如果方格里没有符号，打印1，然后右移：



现在我们来到了m-格局 𝔣 。右移：



机器现在到达了m-格局 𝔟 ，回到了初始状态，并重新开始这样的循环。机器将以这样的方式打印一个由0和1组成的无限序列。

把表中四行的每一行都视为一个指令，这听上去很诱人，而且事实上，图灵后来也采用了这个术语。但是，我们要认识到这些行并不是给机器的指令，它们只是对机器的描述。这就是为什么采用“状态”这一术语更好的原因。如果我们认为这些行是指令，那也就是在暗示我们可以用其他指令来代替它们，从而让同一台机器表现出不同的行为，但那就意味着机器是在解释那些指令，而事实并不是这样（至少现在还不是）。这个机器只执行一个特定的任务。机器真正的工作原理并不重要，重要的是我们可以使用基于格局、符号和操作这些术语的标准方式来表示机器的工作情况。

这台机器能造出来吗？这种特定的机器可以用多种不同的方式制造出来。它可能有一个转轮，转轮的圆周上是一个能交替打印0和1的自动回墨型橡皮印章。创造一台完全按上述方式工作的图灵机——一台真的要去扫描并解读字符的机器——需要的内部逻辑结构恐怕比其外部所展示出来的要复杂得多。图灵机最常见的“制造”方式，则是用计算机来模拟。

图灵机根据扫描符在m-格局之间跳转。这种“条件分支”（计算机科学中很常见）并不是这个时代中的早期计算机所擅长的。康拉德·楚泽利用在35毫米胶片上打孔的方式来编码他的机器指令。在他的第一台机器Z1中，所有的指令必须按顺序执行；Z3可以实现跳转，但是条件分支显得非常笨拙。直到计算机开始在内存中存储程序（“储存程序型计算机”），条件分支才变得简单而常规。

在前面那个表中，符号一列总是“None”，意思是只有方格为空时格局才起作用。如果这台机器碰巧扫描到一个有符号的方格，机器将不知所措。它可能会停住，可能会崩溃，也可能会烧掉，还可能会重新格式化硬盘驱动器。我们不知道会发生什么。不过，无论发生什么，这样的机器将不再被认为是“非循环的”。然而，只要这台机器以一条空白纸带开始，那就不是问题。

图灵通过定义一台可以打印序列

01010101…

的机器，来说明这是一个可计算序列。在这个序列前加一个点可将其转换成一个可计算数：

.01010101…

现在我们就知道了，机器正在计算的是有理数1/3的二进制等价形式。如果你把它们的顺序调换一下（先是1再是0），那么机器将计算下面的二进制数

.10101010…

这实际上是2/3。

下面我所展示的机器将计算1/4，其二进制形式为：

.01000000…

这台机器遵循图灵的约定，使用德文字体b、c、d、e和f来表示m-格局：

格局		行为	
m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{d}$
$\mathfrak{d}$	None	$P1, R$	$\mathfrak{e}$
$\mathfrak{e}$	None	R	$\mathfrak{f}$
$\mathfrak{f}$	None	$P0, R$	$\mathfrak{e}$

特别注意最后两个m-格局 $\mathfrak{e}$ 和 $\mathfrak{f}$ 。它们反复交替，便能让机器打印无限的0。机器必须要不断地打印0，才能符合图灵对于“非循环”的定义。

现在应该非常非常明显，我们可以定义类似的计算机器来计算任何有理数。此处，有理数完全不用考虑。

图灵在之前（第1节的第二段）说：“机器也可以改变正被扫描的方格，但只能移到左边或右边一格。”现在，他想要把这条规定变得稍微灵活点。

[234]

If (contrary to the description in §1) we allow the letters *L*, *R* to appear more than once in the operations column we can simplify the table considerably.

m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0$	$\mathfrak{b}$
	0	$R, R, P1$	$\mathfrak{b}$
	1	$R, R, P0$	$\mathfrak{b}$

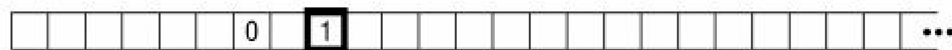
如果（与§1中描述的不同）我们允许字母*L*、*R*在操作列中多次出现，那么我们可以大大地简化表格。

m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$	None	$P0$	$\mathfrak{b}$
	0	$R, R, P1$	$\mathfrak{b}$
	1	$R, R, P0$	$\mathfrak{b}$

（很快，图灵也将允许在一个格局中出现多个*P*操作。）现在，表中只有一个m-格局，所有的操作都依赖于扫描符。如果扫描符为None（只有当机器开始运行时出现），那么机器仅仅打印一个0。



读写头不移动。机器的m-格局仍保持不变，但现在扫描符为0，于是机器就向右移动两格然后打印1：



现在扫描符为1，那么机器向右移动两格然后打印0：



又一次，机器在相间的方格中打印0和1了。

这里，我们有一个重要的收获：任何一个特定的序列都可以由各种不同的机器计算出来。不过，某个以空白纸带开始的自动机总是计算出相同的序列。（当然，我这里指的是自动机，因为选择机允许人为操作干涉，这样就可以创造出不同的序列。图灵在论文中几乎没有考虑选择机。）向图灵的自动机中插入任何不确定的或者随机的因素，或者获得来自“外部世界”的信息（比如时间和日期、经度和纬度、或者一个网页），都是不可能的。

在一个格局中使用多个*L*、*R*和*P*操作可以大大简化机器，但要牢记的是，这些简化的表总是可以转化到之前那个严格的每个状态只允许有一个操作的表。现在看来这仅仅是一个不足为谈差别，但之后它就会变得非常重要。

II. As a slightly more difficult example we can construct a machine to compute the sequence 001011011101111011111…

Ⅱ. 再举个稍微困难点的例子：构造一台机器来计算序列001011011101111011111…

“稍微”困难点？注意图灵现在想要做的是什么。这个序列里包含了越来越长的连续的1，它们之间由0相隔。开始时是一个1，之后是两个1，然后三个1，等等。图灵明显已经厌倦计算有理数了，现在他想处理的是一个无理数，很可能还是一个超越数。

当这台新机器打印连续的1时，它必须用某种办法“记住”之前一轮

中打印了多少个1，然后在这一轮多打印一个。通过来回扫描，这台机器总能访问前一轮的内容，从而能够利用这一信息打印下一轮。图灵究竟是如何做到这一点的，研究起来会非常有意思。

图灵又一次使用德文字体中的小写字母来表示m-格局，这次的字母是o、q、p、f和b。

> The machine is to be capable of five *m*-configurations, viz. “o”, “q”, “p”, “f”, “b” and of printing “ə”, “*x*” “0” “1”. The first three symbols on the tape will be “əə0”; the other figures follow on alternate squares.

> 这台机器具备5个m-格局，即“o”，“q”，“p”，“f”和“b”，并能打印“ə”，“*x*”，“0”，“1”。纸带上的最前面三个符号会是“əə0”，其他数字在后面相间的方格中出现。

这是图灵首次提到在相间的方格中打印数字（0、1，或者其他第一类符号）。假设最左边的符号出现在纸带的左端，则他想要纸带最后变成：

ə	ə	0		0		1		0		1		1		0		1		1		1		0		•••

当然，纸带永远不会在“最后”变成什么，因为机器会永远运转下去。要使它符合“非循环”的要求，它必须永远不停地打印。

在语音学和语言学领域中，字符ə是大家熟悉的中央元音。图灵使用这个中央元音代表程序员们所说的哨兵（sentinel）。在这个例子中，它是一个指明数字边界的特殊字符。只要方格中的扫描符不是中央元音，读写头就一直向左移动，这样机器可以把读写头移到纸带的最开始处。（为什么这里有两个中央元音？这个例子中只需要一个就够了，但是图灵之后构造的机器需要两个中央元音作为哨兵。他在这个例子中加入第二个中央元音可能是为了保证前后一致。）

在第一台样机中，0和1之间的空格是没有用的，而在这个例子里，空格会扮演着非常重要的角色。

On the intermediate squares we never print anything but “*x*”. These letters serve to “keep the place” for us and are erased when we have finished with them.

在中间的方格中，我们只打印*x*，不打印别的东西。它们用来帮我们“记位置”，用完之后就被擦除。

图灵将纸带上的方格分为两类。机器在每隔一格的地方打印0和1。除了“哨兵”字符，这些方格中没有其他的符号。图灵把中间夹着的方格作为某种临时便笺使用。因此，我们现在可以将其分别称为“数字格”（包含0和1）和“非数字格”（包含其他的字符）。（图灵后来把他们称为F格和E格，分别代表figures和erasable。）

We also arrange that in the sequence of figures on alternate squares there shall be no blanks.

我们还要求，相间方格里的数字序列中没有空格。

机器逐步算出0和1的同时，也把它们从左到右顺次打印出来。机器计算出来的每个新数字都被打印到下一个可用的空的数字格中，不会跳过任何数字格。这些限制构成了一整套规则（有些明显表述出来了，有些则是暗示出来的），后来被埃米尔·波斯特称作“图灵规范机器”。[1]这比一般的“图灵机”约束更多。图灵规范机器从不擦除数字格，也不在一个已有数字的数字格上覆盖一个不同的数字。在后文中，这些隐含的规则将变得非常重要。

下面是图灵的格局表，用来计算他定义的无理数。

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$		$P\text{ə}, R, P\text{ə}, R, P0, R, R, P0, L, L$	$\mathfrak{o}$
$\mathfrak{o}$	1	R, Px, L, L, L	$\mathfrak{o}$
	0		$\mathfrak{q}$
$\mathfrak{q}$	Any (0 or 1)	R, R	$\mathfrak{q}$
	None	$P1, L$	$\mathfrak{p}$
$\mathfrak{p}$	x	E, R	$\mathfrak{q}$
	ə	R	$\mathfrak{f}$
	None	L, L	$\mathfrak{p}$
$\mathfrak{f}$	Any	R, R	$\mathfrak{f}$
	None	$P0, L, L$	$\mathfrak{o}$

格局		行为	
m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$		$P_{\text{ə}}, R, P_{\text{ə}}, R, P0, R, R, P0, L, L$	$\mathfrak{o}$
$\mathfrak{o}$	1	R, Px, L, L, L	$\mathfrak{o}$
	0		$\mathfrak{q}$
$\mathfrak{q}$	Any（0 or 1）	R, R	$\mathfrak{q}$
	None	$P1, L$	$\mathfrak{p}$
$\mathfrak{p}$	x	E, R	$\mathfrak{q}$
	ə	R	$\mathfrak{f}$
	None	L, L	$\mathfrak{p}$
$\mathfrak{f}$	Any	R, R	$\mathfrak{f}$
	None	$P0, L, L$	$\mathfrak{o}$

和前面一样，机器以m-格局 $\mathfrak{b}$ 开始。它首先打印两个中央元音和两个0。纸带将变成：



m-格局b执行的是程序员称为初始化的任务，此后机器再也不会进入m-格局b。

在我们翻看机器内部弄得满手是油之前，先来大致体会一下其他m-格局是干什么的。在一些格局（特别是q、p和f）中，操作列展示了一次移动两格的情况：*R*，*R*或者*L*，*L*。在这些情况下，机器能够有效地沿着数字格（q和f的情况）或非数字格（p的情况）移动。

除了b之外，其他m-格局还会在遇到特定的扫描符时转回到自身。程序员通常称这样的操作为循环。循环可以实现重复性的工作，甚至包括寻找某个特定的符号这样简单的操作。

m-格局o在数字格上穿过一连串的1自右向左移动，机器在这个过程中每发现一个1，就会在这个1的右边打印一个*x*，然后向左移动检查下一个数字格。结束时，它将转换到m-格局q。

m-格局q从左至右沿着数字格移动，直到它遇到一个空格。这是当前序列结束的地方。然后它打印一个1，向左移动（至一个非数字格），转到m-格局p。

类似地，m-格局f也沿着数字格向右移动，直到遇到一个空

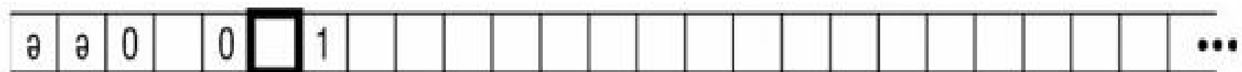
格，然后打印一个0，向左移动两格，再转换到m-格局 $\mathfrak{o}$ 。

m-格局 $\mathfrak{p}$ 相当于一个调度员。绝大多数时间里，它都是在非数字格上向左移动寻找*x*符号。当发现一个*x*，它就将其擦除，向右移动，并转到m-格局 $\mathfrak{q}$ 。如果它遇到一个中央元音便向右移动，并且转换到m-格局 $\mathfrak{f}$ 。

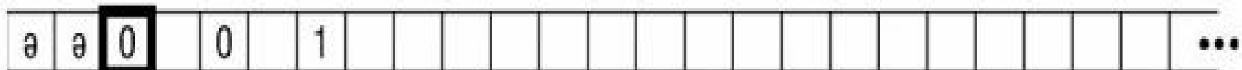
图灵使用符号*x*的方式很聪明。当构造一串新的连续数字1时，机器先在前一串连续数字1中的每一个1后打印一个*x*，随后在已有序列的最后打印一个1，然后每有一个*x*就再打印一个1，从而让串的长度增加了1。

尽管我们可以在每一个操作之后说明纸带的情况，但是对于这个例子，最好还是在每一个格局结束时查看一下纸带。

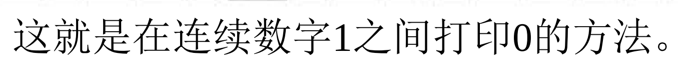
机器从m-格局 $\mathfrak{b}$ 至 $\mathfrak{o}$ 。但是，对于扫描符0，$\mathfrak{o}$ 不做任何事，而是直接转到m-格局 $\mathfrak{q}$ 。对于m-格局 $\mathfrak{q}$ ，如果扫描符是0或1，读写头向右移动两格并仍处于相同的m-格局。但若碰到一个空格，它将在打印一个1后向左移动。总而言之，m-格局 $\mathfrak{q}$ 沿着数字格向右移动，直到它遇到一个空格，然后打印一个1，再向左移动。



下一个m-格局 $\mathfrak{p}$ 一般沿着非数字格移动。它向左移动两格，直到遇到一个包含*x*或中央元音的非数字格。在这个例子中，它遇到的是中央元音，于是向右移动：



m-格局是　　时，读写头沿着数字格移动。它保持每次向右移动两格，直到扫描到空格。这时候它就会打印一个0，然后向左移动两格：



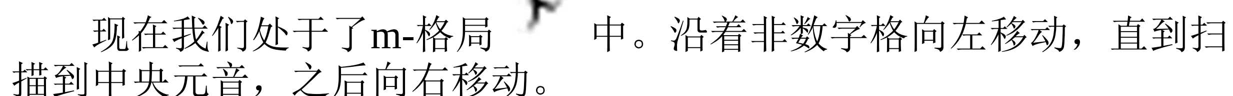
这就是在连续数字1之间打印0的方法。

现在我们来到了m-格局　　。这个m-格局总是开始于一连串数字1中最右边的那个1。它的任务是在每一个1的后面打印一个*x*，在这串连续数字1的左侧的0处停止。

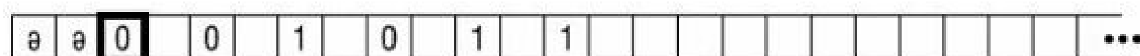
回到m-格局  。这个m-格局沿着数字格向右移动直到遇到空格，在打印一个1后向左移动。

m-格局　　沿着非数字格向左移动，直到遇到一个*x*或者中央元音。当它遇到一个*x*，就擦除这个*x*，并向右移动。

又一次回到m-格局  。沿着数字格向右移动，直到碰到一个空格，在打印完一个1后向左移动。



现在我们处于了m-格局　　中。沿着非数字格向左移动，直到扫描到中央元音，之后向右移动。



m-格局 $\mathfrak{f}$ 在数字格上向右移动，直到它遇到一个空格，它打印完一个0后向左移动两格。

ə	ə	0		0		1		0		1		**1**		0	…

这样看起来确实奏效。我们现在有了连续的一个1和连续的两个1，我们来看一下它能否继续按我们的期望工作下去。

m-格局 $\mathfrak{o}$ 的工作是在最后那串连续数字1的每一个1后打印一个*x*。

ə	ə	0		0		1		**0**		1	x	1	x	0	…

m-格局 $\mathfrak{q}$ 沿着数字格向右移动，直到遇到一个空格。然后，它打印一个1，向左移动。

ə	ə	0		0		1		0		1	x	1	x	0		1	…

注意，现在这里有两个*x*，并且现在这串连续数字1还差两个1。对于每一个要被擦除的*x*，都会有一个1打印出来。m-格局 $\mathfrak{p}$ 沿着非数字格向左移动，直到遇到一个*x*，然后擦除*x*并向右移动。

ə	ə	0		0		1		0		1	x	1		**0**		1	…

m-格局 $\mathfrak{q}$ 沿着数字格向右移动，直到遇到一个空格，打印一个1后向左移动。

ə	ə	0		0		1		0		1	x	1		0		1		1	…

回到m-格局 $\mathfrak{p}$ ，继续向左移动，直到遇到*x*，擦除*x*后向右移动。

ə	ə	0		0		1		0		1		**1**		0		1		1	…

m-格局 $\mathfrak{q}$ 在结尾处再打印一个1。

ə ə 0 0 1 0 1 1 0 1 1 1 …

现在是m-格局 $\mathfrak{p}$，读写头向左移动，直到遇到一个中央元音。

ə ə 0 0 1 0 1 1 0 1 1 1 …

m-格局 $\mathfrak{f}$ 沿着数字格向右移动，直到序列的末端，打印一个0。

ə ə 0 0 1 0 1 1 0 1 1 1 0 …

现在机器已经成功地打印了包含三个1的连续数字串和一个0。

图灵是如何想到这台机器所使用的技巧的呢？我猜他抵住了直接去数的诱惑，而是尝试手动计算这个序列。他可能发现了，可以通过在数字上标注小标记来追踪连续的数字串。这些小标记变成了机器打印在非数字格中的字符*x*。

纸带的示意图并没有出现在图灵的论文中，他没有兴趣为机器或其操作提供这种赤裸的“写实”图示。相反，他有一种别的方法来表示机器的工作状态。

在其论文的第二部分（本书第67页），图灵说：“在机器运转中的任何阶段，被扫描方格的标号、纸带上所有符号构成的序列以及m-格局，共同描述了这个阶段的完全格局。” 尽管图灵提到的“被扫描方格的标号”有点怪，因为方格并没有明确的编号，不过对于只在一个方向无限延伸的纸带，方格有隐含的编号。

图灵将要展示一种用完全格局——即纸带的快照加上当前m-格局和扫描格——来表示机器工作状态的方法。

To illustrate the working of this machine a table is given below of the first few complete configurations. These complete configurations are described by writing down the sequence of symbols which are on the tape,

[235]

with the *m*-configuration written below the scanned symbol. The successive complete configurations are separated by colons.

为了说明机器的运作，我们给出下表，表中包括了最初的几个

完全格局。我们写下纸带上的符号序列，并把m-格局写在扫描符下面，以此来描述这些完全格局。后续的完全格局用冒号隔开。

在论文中，接下来是一个“表”的四个条目。每个条目各有两行，初看起来非常令人费解。下面是第一个条目。

:əə0　0:əə0　0:əə0　0:əə0　0　:əə0　0　1:
𝔟　𝔬　𝔮　𝔮　𝔮　𝔭

:əə0　0:əə0　0:əə0　0:əə0　0　:əə0　0　1:
𝔟　𝔬　𝔮　𝔮　𝔮　𝔭

注意这些冒号！每一对冒号之间的是纸带的连续快照。一些0和0以及（后面）0和1之间的空隙比正常的空格要大一点。这种偏大的空隙代表一个空格。与显示在纸带下的m-格局一起，它们构成了机器的前六个完全格局，显示了目前已经打印在纸带上的所有符号。

第一个 𝔟 表明起始格局。开始时，纸带是空白的。这个格局打印出了最初两个冒号间的序列，和前面显示的一样：

ə	ə	**0**		0																					…

之前使用黑框来表示读写头的位置在下一个扫描符处，现在图灵在下一个扫描符的下面给出了下一个m-格局：

:əə0　0:
𝔬

因为当扫描符是0时，m-格局 𝔬 不做任何操作，所以下一个纸带的快照和之前的一样，但是m-格局已经转换成了 𝔮 ：

:əə0　0:
𝔮

当m-格局 $\mathfrak{q}$ 扫描到一个0时，读写头向右移动两格，下一个m-格局仍然是 $\mathfrak{q}$ ：

```
:əə0   0      :
              q
```

这次的扫描符又是0。读写头向右移动两格， m-格局仍然是 $\mathfrak{q}$ ：

```
:əə0   0      :
              q
```

注意，当读写头超过上次打印的字符后，纸带是如何变得更宽点的。现在，扫描符是空格，因此机器打印一个1，向左移动一格，转换到格局 $\mathfrak{p}$ ：

```
:əə0   0   1:
           p
```

虽然这样做的视觉满足感不及真实的纸带，但是图灵的方法提供了更多的信息，特别是在读写头的当前位置下面指明了下一个m-格局。这些连续的完全格局展现了机器操作的全部历史记录。我们很容易查看其中任何一个完全格局，将m-格局和扫描符与机器的状态匹配起来，最终得到下一个完全格局。

图灵给出的下一个序列展示了m-格局 $\mathfrak{p}$ 往回搜索，直到找到中央元音，然后转换到格局 $\mathfrak{f}$ ，转而向右寻找空格。

```
əə0  0  1:əə0  0  1:əə0  0  1:əə0  0 1:
       p      p           f           f
```

```
əə0  0  1:əə0  0  1:əə0  0  1:əə0  0 1:
     𝔭        𝔭             𝔣              𝔣
```

下一个条目：仍然是m-格局 $\mathfrak{f}$ ，机器找到一个空的数字格（注意冒号之间的空间是如何再次加宽的），打印一个0，向左移动两格后转换到格局 $\mathfrak{o}$ 。

```
əə0  0   1:əə0   0  1    :əə0   0 1  0:
         𝔣               𝔣           𝔬
```



```
əə0  0   1:əə0   0  1    :əə0   0 1  0:
         𝔣               𝔣           𝔬
```


m-格局 $\mathfrak{o}$ 再遇到扫描字符1后向右移动，打印一个*x*后向左移动三格。

```
əə0   0   1x0: ....
      𝔬
```



```
əə0   0   1x0: ....
      𝔬
```


图灵就展示了这么多。如果你觉得这种纸带操作的表示法并不简洁明了，图灵也给出了另外一种方法。

This table could also be written in the form

𝔟:əə𝔢0 0:əə𝔮0 0: ..., (C)

in which a space has been made on the left of the scanned symbol and the *m*-configuration written in this space.

这个表也可以写成

𝔟:əə𝔢0 0:əə𝔮0 0: ..., (C)

在这里，扫描符的左侧留出了一块空隙，m-格局被写在这个空隙里。

图灵使用字母C（代表Configuration，格局）来标识这种格式，他将在第6节中提到。之前展示的完全格局：

:əə0 0:
𝔢

现在可以表示为：

:əə𝔢0 0:

现在，我们可以看出图灵使用德文字体表示m-格局至少有一个原因：以这种格式，m-格局可能没那么容易与机器打印出的符号区别开了。在每一对冒号之间的字符序列将不再同纸带上一模一样，因为需要为下一个m-格局预留一个空格。图灵也承认它有点奇怪。

This form is less easy to follow, but we shall make use of it later for theoretical purposes.

这个形式比较难明白，但出于理论目的，我们后面会用到它。

事实上，使用这种形式是必需的，不过我们还会对此形式再做一下修改。图灵已经着手组装，开始准备后面的一个主要成果展示了：他将展示一个通用计算机器——现在通常称图灵机或者UTM（Universal Turing Machine）——功能上（抛开商业意义不算）等价于现代计算机。

注意最后这个格式的优点：机器的整个操作都被排成一个字符流，这是程序员非常喜爱的形式。当读写文件或者进行数字通信时，理想的

方法是读或写字符流，从头到尾一个接着一个，从不向前或向后跳跃。

我们同样注意到，图灵将下一个m-格局移到了下一个扫描符的前面。图灵将这两项合起来定义为格局，并且这两项在完全格局中出现的顺序，与它们在机器表前两列中出现的顺序一致。你可以把这个m-格局和符号配成一对，并通过扫描机器表的m-格局和符号列来寻找匹配项。（很明显，当机器表中的符号列包含的是实际的字符，而不是“Any”、“None”或空白时，这样的查找更简单。）实际上，图灵会在构造他的通用机时让这个搜索过程实现自动化。

图灵接下来讨论了他选择在相间的方格打印数字序列的理由。

> The convention of writing the figures only on alternate squares is very useful: I shall always make use of it. I shall call the one sequence of alternate squares *F*-squares and the other sequence *E*-squares. The symbols on *E*-squares will be liable to erasure. The symbols on *F*-squares form a continuous sequence. There are no blanks until the end is reached.

> 将数字只写在相间的方格中，这个约定会非常有用，我会一直用到它。我会将其中一个由相间的方格组成的序列称为F-格，另一个这样的序列称为E-格。E-格中的符号可以擦除。F-格中的符号形成一个连续的序列。在到达纸带末端之前，将不会出现空格。

之前我提及这些格时，称它们是数字格和非数字格。只需要记住figures（数字，即0和1）和erasable（可擦除的）这两个单词，你就能分清它们分别是哪个了。“在到达纸带末端之前，将不会出现空格”，这句话只针对F-格。一个可计算序列的数字总是从左至右按次序打印的，从不跳过一个F-格，也不改写F-格中的数字。这些是图灵的通用机所必须遵守的规则。

E-格是一种便笺式存储器，某种程度上可能等价于人类的记忆。

> There is no deed to have more than one *E*-square between each pair of *F*-squares: an apparent need of more *E*-squares can be satisfied by having a sufficiently rich variety of symbols capable of being printed of *E*-squares.

在每对F-格的中间只需要一个E-格：明显需要多个E-格时，可以允许在E-格中打印足够多样的字符。

图灵的第二台机器采用了一种技巧，即通过在E-格中打印*x*符号来实现字符识别。这是他经常采用的一般性技巧，因而给这一技巧命了名。

If a symbol β is on an *F*-square *S* and a symbol α is on the *E*-square next on the right of *S*, then *S* and β will be said to be *marked* with α. The process of printing this α will be called marking β (or *S*) with α.

如果符号β在F-格*S*中，符号α在紧邻*S*右侧的E-格中，那么*S*和β就叫做是用α标记的。打印这个α的过程称为用α来标记β（或*S*）。

下图中的0（在F-格上）称为用*x*标记的：



事实说明，这些标记非常方便，这也是图灵最棒的发明之一。

尽管如此，标记并不是强求的。定义一台只使用两个字符的机器，或者只区分空格和被标记格的机器，都是可能的。数学家埃米尔·波斯特在一篇有意思的论文[2]中探讨过这个方法，这篇论文独立地给出了一种设置，它与图灵定义的类似。波斯特设想有一个“工人”和一些“盒子”，这些盒子被排成了一个序列。这个工人能够完成以下工作：

(a) 标记他所在的盒子（假设盒子为空）；
(b) 擦除他所在盒子的标记（假设盒子已标记）；
(c) 移动到他右侧的盒子里；
(d) 移动到他左侧的盒子里；
(e) 判断他所在的盒子是否被标记。

波斯特其实并没有展示他的工人是如何完成实际应用的。如果方格或者盒子只有被标记和不被标记两种状态，工作起来明显比图灵的捷径费力得多。

---

[1] 在埃米尔·波斯特的论文 “Recursive Unsolvability of a Problem of Thue”，*The Journal of Symbolic Logic*，Vol. 12，No. 1（Mar 1947），1-11的附录中。整篇文章转载于Martin Davis，ed.，*The*

Undecidable（Raven Press，1965），293-303。附录转载于B. Jack Copeland，ed.，*The Essential Turing*（Oxford University Press，2004），97-101。

[2] 埃米尔·波斯特，“Finite Combinatory Processes Formulation I”，*The Journal of Symbolic Logic*，Vol. 1，No. 3（Sep. 1936），103-105。重印于Martin Davis，ed.，*The Undecidable*，289-291。尽管波斯特的论文比图灵的论文发表得早，但是波斯特的论文是 *The Journal of Symbolic Logic* 杂志在1936年10月7日收到的，而 *the Proceedings of the London Mathematical Society* 在1936年5月28日就收到了图灵的论文。

第6章
加　与　乘

早在1935年5月，阿兰·图灵就考虑去普林斯顿大学，也申请了普林斯顿大学的访问奖学金[1]。一年后，他发现普林斯顿大学数学系教授邱奇也发表了一篇关于判定性问题的论文，于是图灵“相当肯定地决定”[2]要去普林斯顿大学。

麦克斯·纽曼为此提供了帮助。纽曼向邱奇介绍了图灵的工作（本书第52页），并在同一封信中，请他帮助图灵获得奖学金：

> 我应该指出，图灵的工作是完全独立进行的，一直没有得到任何人的指导或者评判。因而，让他尽早接触本领域的顶尖人员变得更加重要，这样他才不致于孤独成性。[3]

倾向于独立工作，不受外界影响，这实际上是图灵的一个大问题。早在他年轻的时候，图灵就重新创立了二项式理论，并发明了自己的微积分记号。在尝试解决判定性问题时，他不熟悉邱奇及其同事们的早期成果，这也许是件好事，否则他可能就不会找到这样有趣的解决方法了。然而，一般说来，还是有必要知道在世界其他地方发生了什么事情，而对于数理逻辑领域，普林斯顿就是这样的地方。图灵没能获得他申请的普罗科特奖学金，但得到了国王学院的奖学金。

新泽西州普林斯顿的知识光环由于高等研究院的成立而变得更加熠熠生辉。高等研究院的成立得到路易斯·班伯格5百万美元的捐赠。班伯格创建了班伯格百货连锁店，并在1929年经济大萧条之前将其出售给了梅西百货公司。

高级研究院一开始成立的目的是为了促进科学和历史研究。在最初的几年中，高级研究院的数学学院与普林斯顿大学的数学系在同一座楼，这促成了两个机构之间的许多交流。高级研究院迅速成为了优秀科学家和数学家的家园，他们中的一些人是逃离了危险的欧洲来到这里的，其中最有名的是爱因斯坦。他于1933年来到这里，并在此度过了余生。

图灵于1936年9月到达普林斯顿大学时，非常想见到库尔特·哥德尔。一年前，哥德尔还身在高级研究院，之后也回来过，可惜的是一直未能与图灵谋面。

图灵在剑桥大学时见过的约翰·冯·诺依曼此时在高级研究院，还有同样来自剑桥大学的G. H. 哈代。理查德·柯郎和赫尔曼·外尔也在高级研究院，他们几年前逃离了哥廷根。

图灵在普林斯顿大学待了两年，并获得了第二年的普罗科特奖学金（总共2000美元），邱奇成为了图灵的论文指导教授。在邱奇的指导下，图灵写了一篇论文[4]，并在1938年6月21日获得了博士学位。图灵拒绝了约翰·冯·诺依曼提出的一份1500元年薪、担任其助理的工作，并于一个月后回到了英国。

1939年春，阿兰·图灵回到了剑桥大学，教授数学基础这一课程。四年前，图灵上过麦克斯·纽曼教授的数学基础，并了解了判定性问题。现在，他可以基于自己在可计算数方面的工作，在期末考试中提出关于判定性问题不可解的问题了。[5]

在论文中，图灵试图让读者相信，他的机器确实可以计算非平凡的数字序列。到目前为止，我们还没有真正见过可以称为计算的东西。在图灵的第一个例子中，表面上看，机器能打印出1/3的二进制表示，但它只是通过笨拙地进行0和1的交替来做到这一点的。当然，这不是用1除以3，机器也没有实现用分子除以分母来计算任意有理数的通用过程。

即使是一个从事处理底层机器代码工作的程序员，也会习惯于可以执行加法和减法等基本数学运算的计算机硬件。出于这个原因，我们可能会怀疑（甚至有点担心）一台甚至连加法都要通过自定义格局和操作来完成的机器。

让我们来构造一个非平凡的机器，当面对峙这种担忧。让我们说服自己，图灵机确实可以作加法和乘法运算（当然还有减法、除法和乘方，甚至是写一首诗）。

第一个例子是一个能够依次算出所有正整数的小图灵机。这个图灵机并不符合图灵的约定，因为它写下的每一个新数字都会覆盖掉之前的数字。在打印结果时，它不跳过任何方格，并用下一个数代替每一个结果。此外，考虑到这些数都是整数，我把机器打印数字的方式设计得跟我们平时写数一样——更高的数位往纸带左边打，而不是右边。尽管没有遵守图灵的约定，但这台机器确实展示了如何通过不断加1让数字递增，这至少是我们对现代计算机的一个基本要求。

在我的例子中，我没有用德文字母，而是用粗体写的描述性单词，（在后面的例子中）有时会用短横线来连接多个单词。和图灵一样，我

用“none”来指一个空格。和图灵不一样的是，我用“else”来表明该格局适用于所有其他未明确列出的字符。这个机器从格局**begin**开始，而且只有三个m-格局。

m-格局	符号	操作	最终 m-格局
begin	none	$P0$	**increment**
increment	0	$P1$	**rewind**
	1	$P0, L$	**increment**
	none	$P1$	**rewind**
rewind	none	L	**increment**
	else	R	**rewind**

m-格局**begin**只是打印一个0，然后切换到格局**increment**。m-格局**increment**会读取一位数字。如果读到的是0，那么格局**increment**将它变成1，此时便完成了整个整数的一次递增。如果读到的是1，那么格局**increment**将其更改为0，然后向左移动一位。现在它必须对更高一位的数字做增量操作。m-格局**rewind**将读写头向右移至数字的最低位，准备下一次递增。

一旦你开始编写做算术的图灵机，就会立即明白为什么二进制数字这么方便。下面是一个等价的机器，不过它产生的是所有十进制而非二进制的正整数。

m-格局	符号	操作	最终 m-格局
begin	none	$P0$	**increment**
increment	0	$P1$	**rewind**
	1	$P2$	**rewind**
	2	$P3$	**rewind**
	3	$P4$	**rewind**
	4	$P5$	**rewind**
	5	$P6$	**rewind**
	6	$P7$	**rewind**
	7	$P8$	**rewind**
	8	$P9$	**rewind**
	9	$P0, L$	**increment**
	none	$P1$	**rewind**
rewind	none	L	**increment**
	else	R	**rewind**

你们看出问题在哪了吧。这台机器需要逐一处理十进制中的每一种数字。二进制系统则更简单，因为它的可能性更少。二进制的加法表和乘法表非常非常小：

+	0	1
0	0	1
1	1	10

和

×	0	1
0	0	0
1	0	1

我要将这些加法和乘法的运算规则应用在本章的第二个例子中。这是一个遵守了图灵约定的机器，它将会用二进制计算2的平方根。其实，如果假定二进制小数点在所有数位之前，那么机器计算的是

$$\frac{\sqrt{2}}{2}$$

也就是十进制的0.70710678…。为了让讲解清晰易懂，在描述机器时，我将假设它计算的是 $\sqrt{2}$ 。

这个机器执行的算法每次计算一个二进制位。假设机器已经运行了一段时间，并已确定了前四位。二进制 $\sqrt{2}$ 的前四位是1.011，相当于十进制的 $1\frac{3}{8}$ 或者1.375。下一位是什么？这个机器的策略就是，总是假定下一位是1。要检验这样做是否正确，把1.0111和它自身相乘：

```
     1.0111
    x1.0111
 ------------
      10111
      10111
      10111
      00000
      10111
 ------------
 10.00010001
```

这个结果超过了2，所以假设不正确。第五位应该是0，因此得到的前五位变成了1.0110。我们用同样的方法确定第六位。假设第六位是1，让1.01101乘以自身：

```
    1.01101
   x1.01101
```

101101
101101
101101
000000
101101

1.1111101001

得到的结果小于2，所以假设是正确的。我们得到了前六位：1.01101，用十进制表示就是$1\frac{3}{32}$或者1.40625。

显然，图灵机需要用乘法来计算2的平方根。一般来说，两个多位数相乘需要其中一个数的每一位和另一个数字的每一位相乘。假设一个数有n位，另一个数有m位，那么数字乘数字的总次数是$(n \times m)$。

手算乘法时，我们一般用一个数的其中一位乘以整个另一个数，产生n个或m个局部乘积，然后再将它们加在一起。我将展示的机器以一种稍有不同的方法进行乘法计算——在乘法过程中始终维护一个过程和。每个位与位相乘的结果都将加进这个过程和中。这种特殊的加法有一个很微妙的地方就是，每个位与位相乘的结果一般来说并不是加到过程和的最低位上，而是中间的某个位置。

例如考虑1.01101乘以它自身。其六位数的每一位都必须与自身以及其他5位相乘，所以肯定需要36次位与位的乘法。乘法本身很简单：当1乘以1时，结果是1；其他情况下，结果是0。这个结果要加到过程和的哪一位取决于相乘的两位在数中的位置。如果将右起第三位乘以右起第四位，其结果会加在过程和的右起第六位。（将开始位记作第0位时，这样会更说得通一些：右起第三位变成了第2位，右起第四位变成了第3位，加起来是5，乘积就该加到这个位置上。）

计算2的平方根的二进制表示时，我们总是需要让一个n位数与自身相乘。如果结果有$(2n\text{-}1)$位，就意味着该结果小于2，那么最后一位是1就是正确的；如果结果是$2n$位，那么结果将超过2，所以新的最后一位将是0。 这台机器将利用这个结论来确定每一个新的数位是0还是1。

我即将展示的机器符合图灵的约定，这意味着在计算过程中，打印在F-格上的只能是2的平方根中一个接一个的数字。其他的一切——包括维护乘法的过程和——都是在E-格完成的。

这个图灵机还是从m-格局**begin**开始。它采用@符号而不是中央元音来表示哨兵（在如今的计算机上，这是一个更加简单的符号）。这个图灵机以打印哨兵和数字1开始。

m-格局	符号	操作	最终 m-格局
begin	none	$P@, R, P1$	**new**

这样，机器开始时所做的唯一假设，就是2的平方根大于1但小于2。

这个图灵机一旦准备好计算新的一位，就会回到m-格局**new**。这个格局将读写头移动到最左边的数字上。

new	@	R	**mark-digits**
	else	L	**new**

假设机器已经计算出了前几位，让我们来看看机器此后的行为，这样会更容易理解机器的其余部分。下面是一条已经计算出前三位的纸带，这是1.25的二进制表示。机器接着将在标有问号的格子中打印第四位（我将它称为“未知”位）：

@	1		0		1		?																	…

这个问号标记是为了方便理解，它并没有真正出现在纸带上，也没有被机器所使用。

准备做乘法时，机器会标记各个数位。（回想一下，图灵定义的“标记”指的是在一个数字的右边打印一个符号。）这些*x*标记在这台机器中的用法有点类似图灵的范例II（本书第75页）中的机器。m-格局**mark-digits**将用*x*标记所有已知的位，*z*标记未知的位（我将在稍后解释为什么这么做），并在过程和的最低位上打印一个*r*：

mark-digits	0	R, Px, R	**mark-digits**
	1	R, Px, R	**mark-digits**
	none	R, Pz, R, R, Pr	**find-x**

现在，纸带是这样的：

@	1	x	0	x	1	x	?	z		r														…

*r*是过程和的最低位，我们应该把它看作一个0。下面这一段会为每个*x*打印两个*r*，并在此过程中擦去*x*标记。

m-config	symbol	operations	final m-config
find-x	x	E	**first-r**
	@	N	**find-digits**
	else	L, L	**find-x**
first-r	r	R, R	**last-r**
	else	R, R	**first-r**
last-r	r	R, R	**last-r**
	none	Pr, R, R, Pr	**find-x**

这个纸带现在有一个七位的过程和，代表初始值0000000：

@	1		0		1		?	z		r		r		r		r		r		r		r		…

在过程和中，数位的顺序总是和计算出的数字相反。过程和的最低位在左边。如果未知位是1这个假设是正确的，那么在过程和中初始设定七位就足够了。如果还需要第八位，那么未知的位就是0。

机器中需要自乘的数由已经算好的数（本例中是101）和一个假定为1的新数位组成，所以实际的数是1011。为了记住是哪一位正在乘以其他每一位，机器将用字符*x*、*y*和*z*给数字做标记。在做乘法运算的任何时刻，只有一位标记为*x*，一位标记为*y*并且被*x*标记的数字应当和被*y*标记的数字相乘。如果标记为*x*和*y*的数字正好是同一个，那么我们就使用字符*z*，所以任何标记为*z*的数字都将和自身相乘。

这就是为什么未知位（假设为1）有一个初始标记*z*。第一次的乘法涉及未知位乘以自身。在对后面的格局进行分析时，记住下面这一点将会很有帮助：在乘法运算中，每一位都可能被标记为*x*，每一位都可能被标记为*y*，或者仅有一位标记为*z*。

现在，我们准备开始首次的位与位相乘了。这个机器要么将两个分别标有*x*和*y*的位相乘，要么将标有*z*的位进行自乘。m-格局**find-digits**先追溯到哨兵，然后转到**find-1st-digit**寻找最左边那个标记着*x*、*y*或*z*的数字。

m-config	symbol	operations	final m-config
find-digits	@	R, R	**find-1st-digit**
	else	L, L	**find-digits**
find-1st-digit	x	L	**found-1st-digit**
	y	L	**found-1st-digit**
	z	L	**found-2nd-digit**
	none	R, R	**find-1st-digit**

如果**find-1st-digit**检测到了*x*、*y*或者*z*，它会将读写头置于这一位上。根据不同的标记字母，图灵机将转换到**found-1st-digit**或者**found-**

2nd-digit。

如果第一个被标记的数字是0，那么我们就不需要第二个数字了，因为无论如何结果都将是0。所以我们可以通过**add-zero**将0加到当前累积和中。

$$\textbf{found-1st-digit}\begin{cases}0 & R & \textbf{add-zero}\\ 1 & R,R,R & \textbf{find-2nd-digit}\end{cases}$$

如果第一个数字是1，那么我们必须找到第二位数字。机器会搜索第二个标有*x*或者*y*的位。

$$\textbf{find-2nd-digit}\begin{cases}\mathrm{x} & L & \textbf{found-2nd-digit}\\ \mathrm{y} & L & \textbf{found-2nd-digit}\\ \mathrm{none} & R,R & \textbf{find-2nd-digit}\end{cases}$$

第二位决定了加入过程和的是什么。

$$\textbf{found-2nd-digit}\begin{cases}0 & R & \textbf{add-zero}\\ 1 & R & \textbf{add-one}\\ \mathrm{none} & R & \textbf{add-one}\end{cases}$$

需要注意的是，空的F-格是那个未知位，我们已经假定了它是1。在我们的例子中，标记为*z*的位是未知位，所以**add-one**会给过程和加1。

给过程和加0通常不会对它有什么影响，但是，不管你向过程和加了什么，机器都需要对其做一些维护。

在我描述怎样初始化右边E-格上的过程和时，我曾指出字母*r*表示的是0。字母*s*和*t*表示的也是0，而*u*、*v*、*w*表示的都是1。当位与位相乘的结果加入过程和时，我们需要多个字母来记录这一结果加到了哪一位上。

m-格局**add-zero**会将它第一个找到的*r*变成*s*，或者第一个*u*变成*v*：

$$\textbf{add-zero}\begin{cases}r & Ps & \textbf{add-finished}\\ u & Pv & \textbf{add-finished}\\ \mathrm{else} & R,R & \textbf{add-zero}\end{cases}$$

将*r*（代表0）变成*s*（代表0）以及将*u*（代表1）变成*v*（代表1），可以保证在下次有数字加进过程和时，它会被加到下一位上。

将1加到过程和中需要做的更多。第一个*r*（代表0）变为*v*（代表1），或者第一个*u*（代表1）变为*s*（代表0）。对于后者，还需要进行进位的操作。

add-one	*r*	*Pv*	**add-finished**
	u	*Ps, R, R*	**carry**
	else	*R, R*	**add-one**

如果进位导致有数字写入了空格中，那么过程和将超过2，这样格局就变为了**new-digit-is-zero**：

carry	*r*	*Pu*	**add-finished**
	none	*Pu*	**new-digit-is-zero**
	u	*Pr, R, R*	**carry**

在进行了第一次位与位相乘，并将结果加入过程和中后，纸带变成了：

@	1		0		1		?	z		v		r		r		r		r		r		r		…

注意第一个*r*已经变成了*v*（代表1）。

现在，必须移动*x*、*y*和*z*标记来表示下一对要相乘的位。一般来说，*x*标记会左移一个字符。（回想一下，*z*标记表示*x*和*y*标记在此重合，所以这时候标记*z*会变为*y*标记，并且*x*标记印在左边一位）。但是，当标记*x*到达最后（超过最高位）时，*y*标记会左移一个字符，*x*标记移回最右边的数字。当*y*标记到达最后时，乘法就完成了。

开始时，读写头会移到哨兵点：

add-finished	@	*R, R*	**erase-old-x**
	else	*L, L*	**add-finished**

如果**erase-old-x**找到一个*x*，那么会把它擦去。如果找到一个*z*，就会用*y*取代它。不管是哪种情况，读写头都会移动到左边下一个E-格：

erase-old-x	*x*	*E, L, L*	**print-new-x**
	z	*P*y*, L, L*	**print-new-x**
	else	*R, R*	**erase-old-x**

现在可以打印下一个*x*标记了：

$$\textbf{print-new-x}\begin{cases} @ & R, R & \textbf{erase-old-y} \\ y & Pz & \textbf{find-digits} \\ \text{none} & Px & \textbf{find-digits} \end{cases}$$

我们例子中的纸带现在可以回到**find-digits**步骤，为下一次的位与位相乘做准备了：

@	1		0		1	x	?	y		v		r		r		r		r		r		r		•••

这次位与位相乘后会将另一个1加入过程和，只不过这一次将加在后面一位上，因为它总是会加到最左边的*r*或者*u*上：

@	1		0		1	x	?	y		v		v		r		r		r		r		r		•••

机器接着将标记*x*向左移动一个位置：

@	1		0	x	1		?	y		v		v		r		r		r		r		r		•••

这次位与位相乘的结果是0，过程和的值并不改变，但是最左边的*r*会变为*s*：

@	1		0	x	1		?	y		v		v		s		r		r		r		r		•••

标记*x*再次左移：

@	1	x	0		1		?	y		v		v		s		r		r		r		r		•••

新的位与位相乘的结果使得最左边的*r*变成*v*：

@	1	x	0		1		?	y		v		v		s		v		r		r		r		•••

现在，*x*马上就要移进哨兵的位置了。这种情况就由**erase-old-y**和**print-new-y**处理：

erase-old-y	y	E, L, L	**print-new-y**
	else	R, R	**erase-old-y**
print-new-y	@	R	**new-digit-is-one**
	else	Py, R	**reset-new-x**

值得注意的是，如果标记*y*也要移进哨兵的位置了，整个乘法就结束了，并且过程和没有溢出分配给它的空间。这样我们就知道了，未知位是1。

否则，标记*x*必须重置到数的最低位，也就是未知位：

reset-new-x	none	$R, P\mathrm{x}$	**flag-result-digits**
	else	R, R	**reset-new-x**

例子中，纸带的标记*x*和*y*现在如下所示：

@	1		0		1	y	?	x		v		v		s		v		r		r		r		…

我们还需要做许多事情。下一个位与位相乘的结果需要加到过程和的第二位上。为此，当前累各和的第一个*s*或者*v*要（对应地）变为*t*或者*w*：

flag-result-digits	s	Pt, R, R	**unflag-result-digits**
	v	Pw, R, R	**unflag-result-digits**
	else	R, R	**flag-result-digits**

剩下的*s*和*v*则（对应地）变为*r*和*u*：

unflag-result-digits	s	Pr, R, R	**unflag-result-digits**
	v	Pu, R, R	**unflag-result-digits**
	else	N	**find-digits**

这个过程确保下一个位与位相乘的结果会加到过程和的正确位置。

现在纸带真正做好了进行下次位与位相乘的准备，这个位与位相乘

的结果会加到过程和中第一个*r*或者*u*的地方。

@	1		0		1	y	?	x		w		u		r		u		r		r		r		…

整个乘法将会以两种方式中的一种结束，这两种方式你都已经看到过了。如果机器试图将过程和进位到一个空格中，那么我们知道结果肯定会超过2，未知位一定是0，格局就会变成**new-digit-is-zero**。或者，如果*y*标记下一步该到哨兵的位置，那么整个乘法就完成了，并且过程和没超过2，格局变为**new-digit-is-one**。

这两块基本上是一样的。首先，机器会回到哨兵点：

new-digit-is-zero	@	R	**print-zero-digit**
	else	L	**new-digit-is-zero**

现在，机器可以找到空格，并在那里打印0。在移过所有数字的时候，它可以擦除所有还留着的标记。

print-zero-digit	0	R, E, R	**print-zero-digit**
	1	R, E, R	**print-zero-digit**
	none	$P0, R, R, R$	**cleanup**

类似地，m-格局**new-digit-is-one**会打印一个1，作为新的数位，然后也进入**cleanup**模式：

new-digit-is-one	@	R	**print-one-digit**
	else	L	**new-digit-is-one**
print-one-digit	0	R, E, R	**print-one-digit**
	1	R, E, R	**print-one-digit**
	none	$P1, R, R, R$	**cleanup**

在打印出新的数位后，m-格局**cleanup**移除过程和，然后为计算下一个数位跳转到**new**格局。

cleanup	none	N	**new**
	else	E, R, R	**cleanup**

这样，例子中纸带的第四位就计算出来了，现在可以准备计算第五位了。

@	1		0		1		1		?													…

很明显，图灵机并不是一个对程序员友好的环境。大部分编程语言都会有一个叫做sqrt（或类似）的函数，它可以计算2甚至任何其他数的平方根。

然而，这些平方根函数往往都有精度限制。现在的大部分计算机语言都采用美国电气和电子工程师协会（IEEE）的标准来存储浮点数。一个双精度的浮点数只能存储精确到52位的二进制小数，或者大约15到16位左右的十进制小数。如果你需要一个更加精准的数字，很可能就要自己想办法了（直到最近，才出现了很多更高精度的数学函数）。要想像图灵机那样能执行任意多位数的运算，你会发现你要做的会和我刚刚描述的过程差不了多少。

在一台真实的计算机上，你至少可以很方便地进行加法和乘法运算。如果你眼前的任务在一台图灵机上实现几种不同类型的函数，你可能需要考虑组建一个常用表集，作为积木来搭建一些更为复杂的表。

这正是图灵下一步所做的，虽然他的真正目标是建立一个可以模拟任何其他机器的通用机。

---

[1] 安德鲁·霍奇斯，*Alan Turing: The Enigma*（Simon & Schuster，1983）,95。

[2] 霍奇斯，*Alan Turing*，113。

[3] 同上。

[4] 阿兰·图灵，“Systems of Logic Based on Ordinals”，*Proceedings of the London Mathematical Society*，2nd Series，Volume 45（1939），161-228。重印于阿兰·图灵，*Collected Works of A.M.Turing: Mathematical Logic*（Elsevier，2001），161-228，以及B.Jack Copeland，ed.，*The Essential Turing*（Oxford University Press，2004），146-204。

[5] 霍奇斯，*Alan Turing*，152。

第7章
子 程 序

每个程序员都在几乎所有的编程工作中频繁遇到某些类型的任务。有时候，这些任务是完全相同的；更多情况下，它们是有些许区别的。甚至在我们前面提到的计算2的平方根的机器中，若干m-格局都是相当相似的。例如下面三种m-格局：

new	@	R	**mark-digits**
	else	L	**new**
new-digit-is-zero	@	R	**print-zero-digit**
	else	L	**new-digit-is-zero**
new-digit-is-one	@	R	**print-one-digit**
	else	L	**new-digit-is-one**

在一次循环中，这些m-格局都把读写头向左移动，直到遇到一个哨兵。然后，把读写头向右移动一个位置（在最左边的数字上），接着机器转换到另一个m-格局。

预先确定机器中需要的某些相似的m-格局，并且为那些困难的工作预定义特殊的m-格局，是很有好处的。它可以帮助阐明设定图灵机时用到的策略，从而简化最终的工作。

我们把将读写头移回哨兵的m-格局称为**goto-sentinel**。于是，当写入某个特定机器的状态，并且想要把读写头置于哨兵右侧的数字上时，只需要给出m-格局**goto-sentinel**，而不用重复地阐明如何去做。这样不仅简化了机器描述，而且（从理论上）可以帮助任何查看该机器的人理解它。

我们可以就其自身来定义**goto-sentinel**：

goto-sentinel	@	R	?????
	else	L	**goto-sentinel**

而且马上就会发现了那一连串问号指出的问题。机器在找到哨兵之后，必须转向另一个m-格局，但事实上，只有我们真的使用了**goto-sentinel**，机器才能知道要转向哪一个m-格局。我们需要某种指明最终m-格局的方法，以保持**goto-sentinel**的灵活性。

解决方法是将**goto-sentinel**定义成非常类似数学函数的形式，其中最终m-格局就是这一函数的参数：

$$\textbf{goto-sentinel(A)}\begin{cases} @ & R & \textbf{A} \\ \text{else} & L & \textbf{goto-sentinel(A)} \end{cases}$$

现在，我们可以不再使用**new**、**new-digit-is-zero**和**new-digit-is-one**这些m-格局了。在进行平方根运算的机器运行的最开始，我们不用把**begin**转向**new**，再把**new**转向**mark-digits**，可以指定：

begin none *P*@, *R*, *P*1 **goto - sentinel(mark - digits)** 也不用像下面这样把**carry**定义成转向**new-digit-is-zero**：

$$\textbf{carry}\begin{cases} r & Pu & \textbf{add-finished} \\ \text{none} & Pu & \textbf{new-digit-is-zero} \\ u & \text{Pr}, R, R & \textbf{carry} \end{cases}$$

而是使之指向**goto-sentinel**，然后再转到**print-zero-digit**：

$$\textbf{carry}\begin{cases} r & Pu & \textbf{add-finished} \\ \text{none} & Pu & \textbf{goto-sentinel(print-zero-digit)} \\ u & Pr, R, R & \textbf{carry} \end{cases}$$

说到**print-zero-digit**，你是否发现它与**print-one-digit**相比，除了打印的数位不同，功能上是完全相同的？更好的做法是定义一般的**print-digit**函数，其参数就是要打印的字符：

$$\textbf{print-digit(a)}\begin{cases} 0 & R, E, R & \textbf{print-digit(a)} \\ 1 & R, E, R & \textbf{print-digit(a)} \\ \text{none} & Pa, R, R, R & \textbf{cleanup} \end{cases}$$

注意，最后一行的*Pa*操作表明，被打印的字符是**print-digit**的参数。此时，m-格局**carry**变成了：

$$\textbf{carry}\begin{cases} r & Pu & \textbf{add-finished} \\ \text{none} & Pu & \textbf{goto-sentinel(print-digit(0))} \\ u & Pr, R, R & \textbf{carry} \end{cases}$$

m-格局**print-new-y**（用来检测什么时候应该转向格局**new-digit-is-one**）变为：

$$\textbf{print-new-y}\begin{cases} @ & R & \textbf{goto-sentinel(print-digit(1))} \\ \text{else} & Py, R & \textbf{reset-new-x} \end{cases}$$

现在的程序员很快就会了解这个概念。不同的编程语言对此有着不同的说法，如程序、函数或方法，最一般性的说法是子程序。近几十年来，子程序已成为计算机程序中最通用的结构单元。

阅读本书的程序员在把他们对子程序的理解应用到这些带参数的格局上时，需要谨慎些。这些格局最初主要用来阐明图灵机的结构，并且写起来更容易。这里没有“调用”某一格局或者从一个格局中“返回”概念。

图灵在选定“m-函数”这个相对更好的术语前，把这些带参数的格局称为“骨架表”，并把使用“骨架表”的机器表叫做“缩略表”。

4. *Abbreviated tables.*

There are certain types of process used by nearly all machines, and these, in some machines, are used in many connections. These processes include copying down sequences of symbols, comparing sequences, erasing all symbols of a given form, etc. Where such processes are concerned we can abbreviate the tables for the *m*-configurations considerably by the use of “skeleton tables”. In skeleton tables there appear capital German letters and small Greek letters.

4. 缩略表

有些过程几乎所有的机器都使用，并且在一些机器中，这些过程会在很多情况下使用。这些过程包括复制符号序列、比较序列、删除某一形式的所有符号等。在使用这些过程的地方，我们可以使

用“骨架表”来大大简化表中的m-格局。骨架表里通常使用大写德文字母和小写希腊字母。

令人惊讶的是，大写德文字母比它们相应的小写字母更难读。所幸，图灵只使用了A到E之间的字母，不过借助大号字体来熟悉这些大写字母还是会有帮助的：

A B C D E

$\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{E}$

要特别注意，字母A看上去更像U，也要留意C和E之间的细微差别。图灵在本节中使用的希腊字母是斜体的α、β和γ。

These are of the nature of “variables”. By replacing each capital German letter throughout by an *m*-configuration

[236]

and each small Greek letter by a symbol, we obtain the table for an *m*-configuration.

这些是“变量”的性质。把每一个大写德文字母用一个m-格局代替，每一个小写希腊字母用一个符号代替，我们就可以得到一个关于m-格局的表。

我在例子中使用大写拉丁字母表示m-格局，而图灵使用了大写的德文字母。我使用小写拉丁字母表示一个符号，而图灵使用的都是小写希腊字母。他的例子中通常包含多个参数。

现在的子程序（比如sqrt）保存在称为库的文件中，程序员通过指定名字来使用它们。甚至可以说，整个操作系统，比如Unix、Windows或者Mac操作系统，都主要由可供运行在该操作系统上的应用程序使用的子程序组成。

然而，对图灵来说，骨架表的存在仅仅是为了让其更大的机器更易于构建（从他的角度），并且更易于阅读和理解（从我们的角度）。

The skeleton tables are to be regarded as nothing but abbreviations:

they are not essential. So long as the reader understands how to obtain the complete tables from the skeleton tables, there is no need to give any exact definitions in this connection.

骨架表仅仅是个缩略表，它们并不是必不可少的。只要读者明白了如何从骨架表中得到完整的表，这里就没有必要给出任何明确的定义了。

图灵说，骨架表不是必不可少的，的确如此。如果骨架表只是作为一个有趣的概念而提出，并且只限于论文的这一节，那么很容易被略过。然而，图灵正在为他的通用机创造条件，而通用机广泛采用了本节阐述的骨架表。如果没有这些表，对通用机的说明可能更冗长也更复杂。

因此，了解图灵的最终意图，就更容易理解这些表。正如他会在第7节要论述的，通用机解释了一条包含编码成一系列字母的计算机器的纸带。最左端是中央元音的哨兵。纸带在F-格和E-格间交替变化。E-格依然是可擦除的。在通用机中，F-格包含更多的是字母而不是数字。尽管如此，机器总是顺序地从左往右地打印F-格，且不擦除之前的符号。因此，一行中若出现两个空格，则表明那个点的右边没有F-格。

Let us consider an example:

m-config.	Symbol	Behaviour	Final m-config.
$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$	ə	L	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	not ə	L	$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$
$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	None	R	$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$
$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$
	None	R	$\mathfrak{B}$

From the m-configuration $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ the machine finds the symbol of form α which is farthest to the left (the "first α") and the m-configuration then becomes $\mathfrak{C}$. If there is no α then the m-configuration becomes $\mathfrak{B}$.

我们来看个例子：

m-格局	符号	行为	最终m-格局
$\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$	ə	L	$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$
	not ə	L	$\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$
$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$
	None	R	$\mathfrak{f}_2(\mathfrak{C},\mathfrak{B},\alpha)$
$\mathfrak{f}_2(\mathfrak{C},\mathfrak{B},\alpha)$	α		$\mathfrak{C}$
	not α	R	$\mathfrak{f}_1(\mathfrak{C},\mathfrak{B},\alpha)$
	None	R	$\mathfrak{B}$

在 $\mathfrak{f}(\mathfrak{C},\mathfrak{B},\alpha)$ 这个m-格局中，如果机器找到左边最远的α符号（第一个α），则m-格局转向$\mathfrak{C}$。如果没有α，则m-格局转向$\mathfrak{B}$。

他本可从更简单的例子开始，但是这个例子的优点在于，它可以展示出所有的特性。图灵在表的右边给出了解释。（图灵在定义通用机的时候也把解释放在了表的右边。）

尽管图灵真正定义的是一个名为 $\mathfrak{f}$ 的函数，但是这个函数要求了其他两个函数 $\mathfrak{f}_1$和 $\mathfrak{f}_2$。它们都有三个相同的参数：两个m-格局和一个符号。$\mathfrak{f}$ 把读写头向左移动，直到遇到一个中央元音，此时m-格局转向 $\mathfrak{f}_1$。只要方格内不是α，读写头就向右移。（注意，α

是 $\mathfrak{f}$ 的第三个参数。）如果遇到一个*α*，它转到m-格局 $\mathfrak{C}$ ——$\mathfrak{f}$ 的第一个参数。m-格局 $\mathfrak{f}_1$和 $\mathfrak{f}_2$很相似。它们一起可以有效地找到一行里连续的两个空格。当 $\mathfrak{f}_1$遇到一个空格的时候，它转换成 $\mathfrak{f}_2$。如果接下来的字符不是空格，它又转换回 $\mathfrak{f}_1$。只有当 $\mathfrak{f}_2$再遇到一个空格（这个空格一定是一行中连续的第二个空格）时，则停止，并且转向m-格局 $\mathfrak{B}$，$\mathfrak{B}$ 是 $\mathfrak{f}$ 的第二个参数。在这种情况下是没有*α*的。

因此，$\mathfrak{f}$ 代表了find。如果它找到了*α*，则转向m-格局 $\mathfrak{C}$，读写头将设置在第一个（最左边的）*α*处。如果它找不到*α*，就会转向m-格局 $\mathfrak{C}$。

事实上，这个表里有令人困惑的地方。在两个m-格局 $\mathfrak{f}_1$和 $\mathfrak{f}_2$中，“not *α*”的意思看上去像是“任何不是*α*的非空格”，因为另一个格局考虑到了“None”即空格的情况。然而，第一个m-格局不包含“None”情况，为一致起见，它应该包含这一情况。“None”应该和“not ə”[1]的情况是一样的。

在一个完整机器的表里，应该采用类似下面最终m-格局列中的条目来指示骨架表：

m-格局 符号 操作 最终m-格局

$$\mathfrak{f}(\mathfrak{q}, \mathfrak{r}, x)$$

其中，m-格局 $\mathfrak{q}$ 和 $\mathfrak{r}$ 应该在机器的其他地方定义，*x*应该是机器使用的一个符号。

If we were to replace $\mathfrak{E}$ throughout by $\mathfrak{q}$ (say), $\mathfrak{B}$ by $\mathfrak{r}$, and *α* by *x*, we should have a complete table for the *m*-configuration $\mathfrak{f}(\mathfrak{q}, \mathfrak{r}, x)$.

假设把所有的$\mathfrak{E}$用$\mathfrak{q}$代替，$\mathfrak{B}$用$\mathfrak{r}$代替，*α*用*x*代替，那么我们会得到m-格局$\mathfrak{f}(\mathfrak{q}, \mathfrak{r}, x)$的一个完整表。

在完整机器的上下文中，这个骨架表可以有效地扩展为下表：

m-格局	符号	操作	最终m-格局
$\mathfrak{f}$	ə	L	$\mathfrak{f}_1$
	not ə	L	$\mathfrak{f}$
$\mathfrak{f}_1$	x		$\mathfrak{q}$
	not x	R	$\mathfrak{f}_1$
	None	R	$\mathfrak{f}_2$
$\mathfrak{f}_2$	x		$\mathfrak{q}$
	not x	R	$\mathfrak{f}_1$
	None	R	$\mathfrak{r}$

由于同一个机器可能多次使用函数 $\mathfrak{F}$ ，因此，m-格局 $\mathfrak{F}$ 、$\mathfrak{F}_1$和 $\mathfrak{F}_2$的扩展版在每次使用时都需要不同的名字。

$\mathfrak{F}$ is called an “*m*-configuration function” or “*m*-function”.

$\mathfrak{F}$ 被称作 “m-格局函数”或“m-函数”。

这是一个比“骨架表”更好的名字。一般情况下，我会把它们简称为

函数，希望不会引起读者的困惑。

> The only expressions which are admissible for substitution in an *m*-function are the *m*-configurations and symbols of the machine. These have to be enumerated more or less explicitly: they may include expressions such as $\mathfrak{p}(\mathfrak{e}, x)$; indeed they must if there are any *m*-functions used at all.

m-函数中允许被替换的表达式只有机器的m-格局和符号。或多或少地还是有必要明确地把它们列举出来，它们可能包括$\mathfrak{p}(\mathfrak{e}, x)$这样的表达式。事实上，如果使用了任何m-函数，则一定会包含这样的表达式。

如果已经定义了一个名为$\mathfrak{p}$的m-函数，并且一台机器在它的最终m-格局列中引用到该函数，那么必须将$\mathfrak{p}$作为这台机器的m-格局来考虑。

令图灵焦虑的是，一个m-函数的参数可以是其他m-函数。换句话说，m-函数可以嵌套。（不要担心，你会看到很多这样的例子。）问题来自隐含地允许无限递归。无限递归是指一个函数可以指向本身，或者指向第二个函数，而第二个函数又反过来指向第一个函数。如果允许无限递归，那么一个机器最后会包含无穷个m-格局，而这违背了图灵对计算机器的最初定义。

> If we did not insist on this explicit enumeration, but simply stated that the machine had certain *m*-configurations (enumerated) and all *m*-configurations obtainable by substitution of *m*-configurations in certain *m*-functions, we should usually get an infinity of *m*-configurations; *e.g.*, we might say that the machine was to have the *m*-configuration $\mathfrak{q}$

and all *m*-configurations obtainable by substituting an *m*-configuration for $\mathfrak{C}$ in $\mathfrak{p}(\mathfrak{C})$. Then it would have $\mathfrak{q}, \mathfrak{p}(\mathfrak{q}), \mathfrak{p}(\mathfrak{p}(\mathfrak{q})), \mathfrak{p}(\mathfrak{p}(\mathfrak{p}(\mathfrak{q}))),$ … as *m*-configurations.

如果我们不坚持这样明确地列举，而只是简单地声称这个机器有某些特定的m-格局（并列举出来），以及所有可以通过代换某些特定m-函数中的m-格局得到的m-格局，那么通常情况下，我们会得到无穷个m-格局。例如，假设机器拥有m-格局$\mathfrak{q}$，以及所有可以通过代换$\mathfrak{p}(\mathfrak{C})$里的m-格局$\mathfrak{C}$得到的m-格局，则我们会得到如下无穷个m-格局：$\mathfrak{q}, \mathfrak{p}(\mathfrak{q})$, $\mathfrak{p}(\mathfrak{p}(\mathfrak{q})), \mathfrak{p}(\mathfrak{p}(\mathfrak{p}(\mathfrak{q}))),$ …。

我们必须保证，在机器中代换所有的m-格局之后，得到的仍然是有限数量的m-格局。

Our interpretation rule then is this. We are given the names of the *m*-configurations of the machine, mostly expressed in terms of *m*-functions.

这便是我们的解释规则。已知的机器m-格局的名字大多数以m-函数的形式给出。

图灵又提前考虑到了他的通用机，后面可以看到，大部分通用机的确是按照本节定义的m-函数来表达的。

We are also given skeleton tables. All we want is the complete

table for the *m*-configurations of the machine. This is obtained by repeated substitution in the skeleton tables.

骨架表也给出了。我们需要的是一张机器的m-格局的完整表，它可以通过对骨架表的重复代换操作得到。

或许在这一点上，图灵稍微有点偏执了。通常情况下，我们不需要显式地列举出机器的所有m-格局，而只需要知道有限个m-格局即可。

[237]

Further examples.

(In the explanations the symbol “rarr;” is used to signify “the machine goes into the *m*-configuration…”)

更多例子

（解释中的符号“→”用来表示“机器转向某个m-格局……”）

图灵把出现在骨架表右边，看上去常常很隐秘的描述称为“解释”。这些表的列都杂乱地排列在一起，而且没有列标题。有些表只包含m-格局和最终m-格局，而其他的表包含扫描字符列和操作列，它们必须根据内容的不同而有所区分。

图灵在下一个例子中，给出了一个m-函数作为另一个m-函数的参数出现的情况。

$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	From $\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$ the first α is erased and $\rightarrow \mathfrak{C}$. If there is no $\alpha \rightarrow \mathfrak{B}$.
$\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	E	$\mathfrak{C}$	

$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	经过$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$，如果遇到第一个$\alpha$，则擦除它，并且→$\mathfrak{C}$；如果没有$\alpha$，则→$\mathfrak{B}$。
$\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	E	$\mathfrak{C}$	

$\mathfrak{e}$ 表示“擦除”。这个函数首先使用 $\mathfrak{f}$ 寻找（最左边）第一次出现的α，如果找到了，那么将读写头将置于这个字符上。注意，$\mathfrak{f}$ 的第一个参数是函数 $\mathfrak{e}_1$ 。这意味着如果 $\mathfrak{f}$ 找到了字符α，它将转向 $\mathfrak{e}_1$ ，$\mathfrak{e}_1$ 只是简单地擦除这个字符，然后转向m-格局 $\mathfrak{C}$ 。如果 $\mathfrak{f}$ 没有找到α，那么它将转向m-格局 $\mathfrak{B}$ 。

如果你确实仔细检查了这些情况，而不仅仅是接受图灵关于它们可以运行的说法，那么你可能会质疑为什么 $\mathfrak{e}_1$ 需要这么多的参数。它确实不需要这么多，它可以被更简单地定义为$\mathfrak{e}_1(\mathfrak{C})$ 。

提醒程序员们：你们可能因为知道太多而不能准确地理解m-函数的嵌套。尽管难以抵制 $\mathfrak{e}_1$ 在传递给 $\mathfrak{f}$ 前必须以某种方式“被执行”的想法，但是我们还是要抵制这样的倾向，要把 $\mathfrak{f}$ 的第一个参数看作是给出了 $\mathfrak{f}$ 在找到α后最终转向的目的地。

图灵定义了 $\mathfrak{e}$ 函数的第二版本，它包含两个参数，而不是三个：

$\mathfrak{e}(\mathfrak{B}, \alpha)$	$\mathfrak{e}(\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	From $\mathfrak{e}(\mathfrak{B}, \alpha)$ all letters α are erased and → $\mathfrak{B}$.

$\mathfrak{e}(\mathfrak{B}, \alpha)$	$\mathfrak{e}(\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	经过 $\mathfrak{e}(\mathfrak{B}, \alpha)$，将擦除所有的 α，然后→ $\mathfrak{B}$。

定义两个拥有相同的名字但根据不同数目的参数进行了区分的不同函数，是一种非常高级的编程技术，称为函数过载，这在很多老式程序设计语言中是不允许的。

两参数版本的 $\mathfrak{e}$ 函数调用三参数版本的 $\mathfrak{e}$ 函数来擦除第一个 α，但是注意，前者也可以作为后者的第一个参数。后者成功地找到并且擦除了第一个 α，然后转向前者，前者继续调用后者来擦除下一个 α。这个过程一直持续进行，直到所有的字符 α 被擦除。

现在，图灵有效地利用嵌套和递归表示了重复任务的执行，这种做法是很明智的。

然而，使用两参数 $\mathfrak{e}$ 作为三参数 $\mathfrak{e}$ 的一个参数来执行两参数 $\mathfrak{e}$，这样的做法看上去会让我们最担心出现m-格局无限嵌套的情形。

The last example seems somewhat more difficult to interpret than most. Let us suppose that in the list of *m*-configurations of some

machine there appears $\mathfrak{e}(\mathfrak{b}, x)$ $(= \mathfrak{q},$ say).

最后的例子看上去比其他例子更难解释。假设某个机器的m-格局列表中出现$\mathfrak{e}$($\mathfrak{b}$, *x*)（或许会＝$\mathfrak{q}$）。

m-函数$\mathfrak{e}$只有出现在机器的最终m-格局列中时才能在机器运行过程中发挥作用。如$\mathfrak{e}$($\mathfrak{b}$, *x*)，其中$\mathfrak{b}$是机器中使用的m-格局。现在我们可以说$\mathfrak{e}$，($\mathfrak{b}$, *x*)是机器的另一个m-格局，并且只要我们不使用$\mathfrak{q}$代表机器中任何其他的m-格局，我们就还可以用$\mathfrak{q}$来表示这个新的m-格局。

实质上，通过在机器的最终m-格局列中使用$\mathfrak{e}$($\mathfrak{b}$, *x*)，我们已经给机器增加了另一个状态。图灵用两种不同的形式给出它的定义。

The table is

$\mathfrak{e}(\mathfrak{b}, x)$	$\mathfrak{e}(\mathfrak{e}(\mathfrak{b}, x), \mathfrak{b}, x)$

or

$\mathfrak{q}$	$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x).$

这个表是：

$\mathfrak{e}(\mathfrak{b}, x)$			$\mathfrak{e}(\mathfrak{e}(\mathfrak{b}, x), \mathfrak{b}, x)$

或者

$\mathfrak{q}$			$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$.

（最后一行末尾的句号是以“这个表是……”开头的句子的一部分。）这个表暗示 $\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$是机器的另一个m-格局。从下面的推广可以看出，$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$同样也是机器的另一个m-格局。

Or, in greater detail:

$\mathfrak{q}$			$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$
$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$			$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x), \mathfrak{b}, x)$
$\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$		E	$\mathfrak{q}$.

或者，更详细地表示为：

$\mathfrak{q}$			$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$
$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$			$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x), \mathfrak{b}, x)$
$\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$		E	$\mathfrak{q}$.

（同样，最后一行 $\mathfrak{q}$ 之后的句号表示这个表被当成了句子的一部分。）注意，擦除字符后，$\mathfrak{e}_1$ 转向 $\mathfrak{q}$，$\mathfrak{q}$ 已经是这台机器的一个m-格局了，因此，不会导致m-格局的无限产生。

In this we could replace $\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$ by $\mathfrak{q}'$ and then give the table for $\mathfrak{f}$ (with the right substitutions) ad eventually reach a table in which no *m*-functions appeared.

这里，我们使用 $\mathfrak{q}'$ 代替 $\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$，然后给出 $\mathfrak{f}$ 的表（进行适当地代换），最终可以得到一个不包含任何m-函数的表。

正如图灵使用 $\mathfrak{q}$ 来代表格局 $\mathfrak{e}(\mathfrak{b}, x)$一样，他还可以使用 $\mathfrak{q}'$ 表示$\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$, 使用其他的格局来表示 $\mathfrak{e}_1$ 和 $\mathfrak{f}$ 。

现在，我们已经掌握了这些函数（才怪！），图灵不断地积累这些函数。我知道，现在我们很难看出这些函数是如何恰当地搭配在一起的。为了构造通用机，图灵需要一些操作单个字符或字符串的通用类型的函数。读者已经看到了查找和擦除函数。他还需要剪切、复制、粘贴和一些标准打印函数。

函数 $\mathfrak{pe}$ 表示“在结尾打印”。它在第一个空的F-格处打印*β*表示的符号。

$\mathfrak{pe}(\mathfrak{C}, \beta)$			$\mathfrak{f}(\mathfrak{pe}_1(\mathfrak{C}, \beta), \mathfrak{C}, ə)$	From $\mathfrak{pe}(\mathfrak{C}, \beta)$ the machine prints β at the end of the sequence of symbols and $\rightarrow \mathfrak{C}$.
$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	Any	R, R	$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	
	None	$P\beta$	$\mathfrak{C}$	

$\mathfrak{pe}(\mathfrak{C}, \beta)$			$\mathfrak{f}(\mathfrak{pe}_1(\mathfrak{C}, \beta), \mathfrak{C}, ə)$	经过 $\mathfrak{pe}(\mathfrak{C}, \beta)$，机器在符号序列的末尾打印$\beta$，然后→$\mathfrak{C}$。
$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	Any	R, R	$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	
	None	$P\beta$	$\mathfrak{C}$	

这个函数隐藏着一些假设。通常情况下，$\mathfrak{f}$ 寻找其第三个参数在最左边的第一次出现，但是这里的 $\mathfrak{f}$ 第三个参数是一个中央元音，它也是 $\mathfrak{f}$ 为到达序列最左边所要寻找的符号。于是，正如图灵在第85页给出的第二个机器的例子，函数$\mathfrak{pe}$也假设每行中存在两个中央元音。这个m-函数 $\mathfrak{f}$ 首先找到这两个中央元音中最右边的一个（出现在E-格的那个），然后把读写头左移置于左边F-格的中央元音处。随后，函数 $\mathfrak{pe}_1$ 沿着F-格向右移动，直到找到一个空格，并在此打印一个β字符，对于大多数计算机器来说，这个β将是0或1。

接下来的例子都很有技巧。图灵首先定义了两个名字分别为 $\mathfrak{l}$（指左边的）和 $\mathfrak{r}$（指右边的）的函数，然后分别用它们与 $\mathfrak{f}$ 结合得到另外两个函数 $\mathfrak{f}'$ 和 $\mathfrak{f}''$。这两个新的函数在找到所需字符后会将读写头向左或者向右移动。

$\mathfrak{l}(\mathfrak{C})$	L	$\mathfrak{C}$	From $\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$ it does the same as for $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ but moves to the left before → $\mathfrak{C}$.
$\mathfrak{r}(\mathfrak{C})$	R	$\mathfrak{C}$	
$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}\big(\mathfrak{l}(\mathfrak{C}), \mathfrak{B}, \alpha\big)$	
$\mathfrak{f}''(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}\big(\mathfrak{r}(\mathfrak{C}), \mathfrak{B}, \alpha\big)$	

$\mathfrak{l}(\mathfrak{C})$	L	$\mathfrak{C}$	$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$与$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$类似，不同的是它在→$\mathfrak{C}$之前需要向左移动。
$\mathfrak{r}(\mathfrak{C})$	R	$\mathfrak{C}$	
$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}\big(\mathfrak{l}(\mathfrak{C}), \mathfrak{B}, \alpha\big)$	
$\mathfrak{f}''(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}\big(\mathfrak{r}(\mathfrak{C}), \mathfrak{B}, \alpha\big)$	

我本应该把它们称为 $\mathfrak{fl}$ 和 $\mathfrak{fr}$，而非 $\mathfrak{f}'$ 和 $\mathfrak{f}''$，但那只是我的想法。

通用机会要求把字符从纸带的一个位置移动到另一个位置上。函数 $\mathfrak{c}$ 的功能是“复制”。字符α更可能是一个标记。$\mathfrak{c}$ 首先找到这个标记左边的F-格里的字符，然后使用 $\mathfrak{pe}$ 将该字符复制到尾部第一个空的F-格中。

$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}'\big(\mathfrak{c}_1(\mathfrak{C}), \mathfrak{B}, \alpha\big)$	$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$. The machine writes at the end the first symbol marked α and → $\mathfrak{C}$.
$\mathfrak{c}_1(\mathfrak{C})$	β	$\mathfrak{pe}(\mathfrak{C}, \beta)$	

$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}'(\mathfrak{c}_1(\mathfrak{C}), \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$，表示机器在尾部写下由$\alpha$标记的第一个符号，然后→$\mathfrak{C}$。
$\mathfrak{c}_1(\mathfrak{C})$	β	$\mathfrak{pe}(\mathfrak{C}, \beta)$	

注意上面的函数使用$\mathfrak{f}'$来寻找字符α，因此操作结束的时候读写头位于标记的左边，正好是这个标记指示的字符处。

$\mathfrak{c}_1$函数的语法比较独特：扫描到的字符成为了$\mathfrak{pe}$的第二个参数。图灵说：

[238]

The last line stands for the totality of lines obtainable from it by replacing β by any symbol which may occur on the tape of the machine concerned.

最后一行代表了用机器纸带上可能出现的任意符号代替β后得到的所有的行。

例如，如果函数$\mathfrak{c}$只用来复制0或1，那么事实上，$\mathfrak{c}_1$会被定义成：

$\mathfrak{c}_1(\mathfrak{C})$	0	$\mathfrak{pe}(\mathfrak{C}, 0)$
	1	$\mathfrak{pe}(\mathfrak{C}, 1)$

函数$\mathfrak{ce}$代表了“复制并擦除”。它也有分别包含两参数和三参数

的版本。

$\mathfrak{ce}(\mathfrak{E}, \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{e}(\mathfrak{E}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	$\mathfrak{ce}(\mathfrak{B}, \alpha)$. The machine copies down in order at the end all symbols marked α and erases the letters α; $\rightarrow \mathfrak{B}$.
$\mathfrak{ce}(\mathfrak{B}, \alpha)$	$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	

$\mathfrak{ce}(\mathfrak{E}, \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{e}(\mathfrak{E}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	$\mathfrak{ce}(\mathfrak{B}, \alpha)$，表示机器从尾部按顺序复制所有用$\alpha$标记的符号，然后擦除这些$\alpha$，
$\mathfrak{ce}(\mathfrak{B}, \alpha)$	$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	

带三个参数的 $\mathfrak{ce}$ 首先使用 $\mathfrak{c}$ 复制最左边被α标记的数字，然后使用 $\mathfrak{e}$ 擦除这个标记。带两个参数的 $\mathfrak{ce}$ 则使用三参数版本的函数来复制第一个数字并擦除标记，然后再转向两参数版本的函数。实际上，所有标记为α的符号都被复制到纸带尾部第一批可用的F-格里。（在第77页图灵的第二个例子中，本可以使用这个函数把一连串的1复制到纸带的尾部。）

是时候提出效率这个令人生畏的问题了。图灵定义的函数看上去很精密紧凑，但事实上里面隐藏着巨大的工作量。为了执行一次复制和擦除，函数 $\mathfrak{c}$ 要使用 $\mathfrak{f}$ 来寻找标记（并记住 $\mathfrak{f}$ 要一路回溯到哨兵），然后再使用 $\mathfrak{e}$ ，而 $\mathfrak{e}$ 又要调用 $\mathfrak{f}$ 来找到相同的标记后才

能将其擦除。一种更高效的方法是使用 $\mathfrak{ce}$ 在第一次找到某个标记，并在复制字符之前就将其擦除。（为了纪念图灵机臭名昭著的低性能，我们使用“图灵沥青坑”[2]一词来描述过度一般化的计算机例程，准备这些例程比运行它们花费的时间更多。）

但是，图灵并不关心效率问题，毕竟，机器都是想象出来的。如果他愿意，可以以10^6 ZHz[3]的频率来运行机器，没有人会考虑做了多少无用工。

函数 $\mathfrak{re}$ 是指“替换”。假设参数α和β都是标记。函数找到最左边的α，并用β替换它。（因为图灵不允许替换已经在F-格中标记的字符，因此我们可知α和β是标记。）

$\mathfrak{re}(\mathfrak{E}, \mathfrak{B}, \alpha, \beta)$		$\mathfrak{f}(\mathfrak{re}_1(\mathfrak{E}, \mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha)$	$\mathfrak{re}(\mathfrak{E}, \mathfrak{B}, \alpha, \beta)$. The machine replaces the first α by β and $\rightarrow \mathfrak{E} \rightarrow \mathfrak{B}$ if there is no α.
$\mathfrak{re}_1(\mathfrak{E}, \mathfrak{B}, \alpha, \beta)$	$E, P\beta$	$\mathfrak{E}$	

$\mathfrak{re}(\mathfrak{E}, \mathfrak{B}, \alpha, \beta)$		$\mathfrak{f}(\mathfrak{re}_1(\mathfrak{E}, \mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha)$	$\mathfrak{re}(\mathfrak{E}, \mathfrak{B}, \alpha, \beta)$，机器用$\beta$来替换第一个$\alpha$并$\rightarrow \mathfrak{E}$，如果没有$\alpha$，则$\rightarrow \mathfrak{B}$。
$\mathfrak{re}_1(\mathfrak{E}, \mathfrak{B}, \alpha, \beta)$	$E, P\beta$	$\mathfrak{E}$	

三参数版本的函数用β替换所有的α标记。

$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha, \beta)$	$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$. The machine replaces all letters α by β; $\rightarrow \mathfrak{B}$.

$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha, \beta)$	$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$，机器用$\beta$替代所有的$\alpha$字符，然后→$\mathfrak{B}$。

为了保持一致，右边的解释部分的第一行应该缩进两个字符。

如果你已经熟悉了图灵的方法和命名规则，就会知道下面这个函数 $\mathfrak{ce}$ 是用来完成“复制和替换”的。

$\mathfrak{cr}(\mathfrak{C}, \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \alpha), \mathfrak{B}, \alpha)$	$\mathfrak{cr}(\mathfrak{B}, \alpha)$ differs from $\mathfrak{ce}(\mathfrak{B}, \alpha)$ only in that the letters α are not erased. The *m*-configuration $\mathfrak{cr}(\mathfrak{B}, \alpha)$ is taken up when no letters “α” are on the tape.
$\mathfrak{cr}(\mathfrak{B}, \alpha)$	$\mathfrak{cr}(\mathfrak{cr}(\mathfrak{B}, \alpha), \mathfrak{re}(\mathfrak{B}, \alpha, \alpha), \alpha)$	

$\mathfrak{cr}(\mathfrak{C}, \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \alpha), \mathfrak{B}, \alpha)$	$\mathfrak{cr}(\mathfrak{B}, \alpha)$与$\mathfrak{ce}(\mathfrak{B}, \alpha)$的区别仅在于，前者的$\alpha$字母不被擦除。当纸带上没有字母$\alpha$时，m-格局$\mathfrak{cr}(\mathfrak{B}, \alpha)$就被转走。
$\mathfrak{cr}(\mathfrak{B}, \alpha)$	$\mathfrak{cr}(\mathfrak{cr}(\mathfrak{B}, \alpha), \mathfrak{re}(\mathfrak{B}, \alpha, \alpha), \alpha)$	

图灵论文的其他地方没有使用到这些函数。

通用机要求有“搜索并替换”功能，于是图灵接着用了半页的篇幅展示了 $\mathfrak{cp}$ （“比较”）和 $\mathfrak{cpe}$ （“比较并擦除”）函数。由于这些函数里的

最终m-格局太长，因而图灵的解释没有放在每个表的右边，而是放在了下面。（第一行有个错误：最终m-格局列中 $\mathfrak{C}$ 的下标1应该是逗号。在该列出现的一些句号也没有任何意义。）

$\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$		$\mathfrak{f}'(\mathfrak{cp}_1(\mathfrak{C}_1\ \mathfrak{A}, \beta), \mathfrak{f}(\mathfrak{A}, \mathfrak{E}, \beta), \alpha)$
$\mathfrak{cp}_1(\mathfrak{C}, \mathfrak{A}, \beta)$	γ	$\mathfrak{f}'(\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma), \mathfrak{A}, \beta)$
$\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma)$	γ	$\mathfrak{C}$
	not γ	$\mathfrak{A}$.

The first symbol marked *alpha;* and the first marked β are compared. If there is neither α nor β, → $\mathfrak{E}$. If there are both and the symbols are alike, → $\mathfrak{C}$. Otherwise → $\mathfrak{A}$.

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$	$\mathfrak{cp}(\mathfrak{e}(\mathfrak{e}(\mathfrak{C}, \mathfrak{C}, \beta), \mathfrak{C}, \alpha), \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E},$ differs from $\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E},$ $\alpha, \beta)$ in that in the case when there is similarity the first α and β are erased.

$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta)$	$\mathfrak{cpe}(\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta), \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$.

$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E},$ $\alpha, \beta)$. The sequence of symbols marked α is compared with the sequence marked β → $\mathfrak{E}$ if they are similar. Otherwise → $\mathfrak{A}$. Some of the symbols α and β are erased.

$\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$		$\mathfrak{f}'(\mathfrak{cp}_1(\mathfrak{C}_1\ \mathfrak{A}, \beta), \mathfrak{f}(\mathfrak{A}, \mathfrak{E}, \beta), \alpha)$
$\mathfrak{cp}_1(\mathfrak{C}, \mathfrak{A}, \beta)$	γ	$\mathfrak{f}'(\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma), \mathfrak{A}, \beta)$
$\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma)$	γ	$\mathfrak{C}$
	not γ	$\mathfrak{A}$.

比较第一个α标记的字符和第一个β标记的字符。如果即没有标记α也没有标记β，则→ $\mathfrak{C}$。如果标记α和β都存在，并且所标记的字符一样，那么→ $\mathfrak{C}$；否则→ $\mathfrak{A}$。

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$	$\mathfrak{cp}(\mathfrak{e}(\mathfrak{e}(\mathfrak{C}, \mathfrak{C}, \beta), \mathfrak{C}, \alpha), \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$与$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ α, β)的不同之处在于，如果存在一对相同的字符，那么第一个α和β将被擦除。

$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta)$	$\mathfrak{cpe}(\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta), \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E},$ α, β)，比较α标记的符号序列和β标记的符号序列。如果它们是相似的，则→ $\mathfrak{C}$；否则→ $\mathfrak{A}$。一些α和β将被擦除。

图灵在这里所说的“相似”是指“完全相同”。

现在，图灵已经用完了他能想到的容易记忆的所有函数名称，因为他把接下来的函数仅简单命名为 $\mathfrak{q}$ 。不幸的是，这也是一般情况下他用来表示m-格局的字母；更糟糕的是，后文中他使用 $\mathfrak{g}$ 来指代这个函数。

我相信，图灵本意是想用 $\mathfrak{g}$ 来命名这个函数，而不是用 $\mathfrak{q}$ 。类似函数 $\mathfrak{f}$ 用来寻找某个符号（最左边）的第一次出现，这个函数是用来寻找某个符号（最右边）的最后一次出现的。用两个连续的字母来表示有关联的两个 $\mathfrak{f}$ 和 $\mathfrak{g}$ 函数是有意义的，因此，尽管接下来的表里描述的是函数 $\mathfrak{q}$ ，我将使用 $\mathfrak{g}$ 来指代该函数。

只有一个参数的函数 $\mathfrak{g}$ 将读写头右移，直到它在一行中找到两个空格。我们假设那里处于纸带的最右端。带两个参数的函数 $\mathfrak{g}$ 首先使用只有一个参数的 $\mathfrak{g}$ ，然后再向左移动寻找字符α。

[239]

$\mathfrak{q}(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	$\mathfrak{q}(\mathfrak{C}, \alpha)$. The machine finds the last symbol of form α. → $\mathfrak{C}$.
	None	R	$\mathfrak{q}_1(\mathfrak{C})$	
$\mathfrak{q}_1(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	
	None		$\mathfrak{C}$	
$\mathfrak{q}(\mathfrak{C}, \alpha)$			$\mathfrak{q}(\mathfrak{q}_1(\mathfrak{C}, \alpha))$	
$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	α		$\mathfrak{C}$	
	not α	L	$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	

$\mathfrak{q}(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	$\mathfrak{q}(\mathfrak{C},\ \alpha)$，机器找到由$\alpha$标记的最后一个符号，然后→$\mathfrak{C}$。
	None	R	$\mathfrak{q}_1(\mathfrak{C})$	
$\mathfrak{q}_1(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	
	None		$\mathfrak{C}$	
$\mathfrak{q}(\mathfrak{C}, \alpha)$			$\mathfrak{q}(\mathfrak{q}_1(\mathfrak{C}, \alpha))$	
$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	α		$\mathfrak{C}$	
	not α	L	$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	

在本节的最后，图灵给出了一些名字很熟悉的辅助函数。

回想一下，函数 $\mathfrak{pe}$ 用来在最后一个F-格中打印一个字符。函数 $\mathfrak{pe}$ 2在最后两个F-格中打印两个字符。

$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$	$\mathfrak{pe}(\mathfrak{pe}(\mathfrak{C}, \beta), \alpha)$	$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$. The machine prints α β at the end.

$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$	$\mathfrak{pe}(\mathfrak{pe}(\mathfrak{C}, \beta), \alpha)$	$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$，机器在纸带尾部打印$\alpha$和$\beta$。

类似地，函数 $\mathfrak{ce}$ 用来将α标记的字符复制到尾部。函数 $\mathfrak{ce}$ 2则复制α和β标记的字符，函数 $\mathfrak{ce}$ 3复制α、β和γ标记的字符。

$\mathfrak{ce}_2(\mathfrak{B}, \alpha, \beta)$	$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \beta), \alpha)$	$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$. The machine copies down at the end first the symbols marked α, then those marked β, and finally those marked γ; it erases the symbols α, β, γ.
$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$	$\mathfrak{ce}(\mathfrak{ce}_2(\mathfrak{B}, \beta, \gamma), \alpha)$	

$\mathfrak{ce}_2(\mathfrak{B}, \alpha, \beta)$	$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \beta), \alpha)$	$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$，机器在尾部首先复制用$\alpha$标记的字符，接着是用$\beta$标记的字符，最后复制$\gamma$标记的字符。之后，这些$\alpha$、$\beta$和$\gamma$标记被擦除。
$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$	$\mathfrak{ce}(\mathfrak{ce}_2(\mathfrak{B}, \beta, \gamma), \alpha)$	

这些复制是顺序执行的：首先复制所有α标记的字符，其次复制所有β标记的字符，以此类推。图灵使用了带6个参数的函数 $\mathfrak{ce}_5$，这个函数虽然之前没有描述过，但是它的操作是显而易见的。

最后，只有一个参数的函数 $\mathfrak{e}$ 擦除所有的标记。

$\mathfrak{e}(\mathfrak{C})$	ə	R	$\mathfrak{e}_1(\mathfrak{C})$	From $\mathfrak{e}(\mathfrak{C})$ the marks are erased from all marked symbols. → $\mathfrak{C}$.
	Not ə	L	$\mathfrak{e}(\mathfrak{C})$	
$\mathfrak{e}_1(\mathfrak{C})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{C})$	
	None		$\mathfrak{C}$	

$\mathfrak{e}(\mathfrak{C})$	ə	R	$\mathfrak{e}_1(\mathfrak{C})$	经过$\mathfrak{e}(\mathfrak{C})$，所有的标记将从被标记的字符上擦除，然后→$\mathfrak{C}$。
	Not ə	L	$\mathfrak{e}(\mathfrak{C})$	
$\mathfrak{e}_1(\mathfrak{C})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{C})$	
	None		$\mathfrak{C}$	

年长点的程序员可能知道一本由Pascal语言的创始人尼克劳斯·威茨（1934—）编著的书，书名非常美妙，叫作《算法+数据结构＝程序》（Prentice-Hall，1975年出版）。顾名思义，一段计算机程序包括代码（算法）和代码处理的数据。至此，图灵已经展示了他的通用机所需的诸多算法，但是还没有描述如何让一台计算机器来成功地处理数据。这就是接下来要讲的内容。

---

[1] 这是埃米尔·波斯特论文的附录中标识在脚注中的订正之一，详见“Recursive Unsolvability of a Problem of Thue”，*The Journal of Symbolic Logic*，Vol. 12，No. 1（Mar. 1947），1-11。整篇文章转载于Martin Davis，ed.，*The Undecidable*（Raven Press，1965），293-303。附录（包括添加的脚注）转载于B. Jack Copeland，ed.，*The Essential Turing*（Oxford University Press，2004），97-101。

[2] tar-pit这本是地质学术语，此处用来指运行冗长缓慢的计算机例程。——编者注

[3] 频率单位，相当于10^{21}Hz。——编者注

第8章 万物皆数字

在如今这个数字化时代，我们已经习惯于将各种形式的信息都表示成数字。文本、图画、照片、声音、音乐、电影，万物都可以进入数字化工厂，以0和1构成的更为复杂的形式保存在计算机和其他设备上。

但是，在20世纪30年代，只有数才算是数字，如果某个人把文本转换成数字，其目的是欺骗和耍阴谋。

1937年秋，阿兰·图灵开始了他在普林斯顿第二年的生活，当时英国和德国之间很快就要爆发一场战争，普林斯顿处于高度恐慌中。当然，他那时正忙于博士论文，也已经对密码学有了兴趣。密码学是涉及科学和数学领域的学科，它创建保密码（密码学）并破解他人代码（密码分析学）。[1]图灵坚信，战争期间，加密信息的最好方式是将单词转换成二进制数字，然后再乘上很大的数字。在不知道那个大数字的情况下，解密信息会涉及很困难的因式分解问题。图灵的这种想法很有先见之明，因为如今大多数这种计算机加密的工作就是这样的。

与大多数数学家不同的是，图灵喜欢亲自动手做事情。为了实现自动的编码机器，他用电磁式继电器制作了一个二进制乘法器。在人们证实电子管足够可靠之前，电磁式继电器是计算机的基本构件。图灵甚至到机械工厂亲自制造继电器，亲手缠绕电磁铁。

当时，德国的陆军和海军已经在使用一种完全不同的加密设备了。一位名叫亚瑟·谢尔比乌斯（1878—1929）的德国电气工程师发明了恩尼格玛密码机（Enigma）。1918年，谢尔比乌斯试图说服德国海军使用这个机器，但是失败了。1923年，恩尼格玛密码机用于商业用途并出售。之后，德国海军很快对它产生了兴趣，最终其他军队也相继开始使用它了。[2]

恩尼格玛密码机有一个类似打字机的26个按键的基本键盘，但没有数字、标点和移位键。键盘上方是以同样形式排列的26个灯泡。信息通过键盘输入时就会被加密。每按下一个字母，另一个不同字母的灯泡会亮起来。这些亮起的字母被手工抄下来，然后发送到收件人那里。（加密的信息可以由人亲自传送，或者通过邮件发送。再后来，人们使用摩

尔斯电码通过无线电传输加密信息。）接收信息的人也拥有一台恩尼格玛密码机，他会在键盘上输入加密信息，闪光的灯即可显示出原始的文本。

键盘按键通过一系列转子与灯泡实现电气连接。每个转子都是每边有26个触头的小圆盘，这些触头代表了字母表里的26个字母。在转子里面，这些触头是对称连接的。比如，如果左边的触头A与右边的触头T连接，那么左边的触头T就将和右边的触头A连接。正是这种对称的连接关系，使得机器既用来加密，也用来解密。

标准的恩尼格玛密码机有三个连接的转子，它们的布线都不一样，每一个都可以设置到26个位置中的一个。加密机器和解密机器上的这三个转子的布线必须完全一样。依照只有恩尼格玛密码机操作员知道的键列表，这些由三个字母组成的设置转子的键可以每天改变。

至此，我对恩尼格玛密码机的所有描述都只能是让它简单地替换字母，这种加密方式即便是最业余的密码破译者也可以轻易破解。它甚至比大部分其他通过字母替换来加密的方式都更简单，因为它内部的连接是完全对称的。也就是说，如果D被加密成S，那么S也相应地被加密成D。

但事实并非如此。每当恩尼格玛密码机的用户在键盘上按下键，转子都会移动。每按下一个键，第一个转子都会向前移动一个位置。比如，如果键入一个包含26个字母A的字符串，那么随着转子在26个位置上的移动，每个A都会被编码成不同的字母。当第一个转子完成了一圈完整的变化后，第二个转子就会向前移动一个位置。接下来，另一个包含26个A的字符串又会被编码成另一串不同的字母。同样地，当第二个转子完成一个循环后，第三个转子又会向前移动一个位置。第四个固定的转子会使电信号按相反方向再次经过这三个转子。只有进行了17576次按键（26的三次方）后，加密模式才会重复。

问题不仅在于此，这些转子是可替换的。机器配备了五个不同的转子，每个转子都可以随意放置在三个转子空位中的任何一个。另一个加强方法是加上一个插接板，这又增加了一层字母混杂。

1932年，三位波兰数学家开始寻找破解恩尼格玛信息的方法。[3]他们认定需要构造可以自动模拟恩尼格玛编码的设备。第一台设备bomb（他们这样称呼）在1938年开始运行，其工作方式就是搜索所有可能的转子设置。其中一名数学家是马里安·雷耶夫斯基（1905—1980），毕业后他在哥廷根度过了一年。他写道，由于“找不到更好的名字”[4]，他们把那些机器叫做bomb，不过也有可能是因为机器发出的嘀哒声而得此名，也或许是用这些数学家们喜欢的某种冰淇淋圣代命名的呢。[5]

英国政府传统上是雇用一些古典文学学者来从事解码工作，因为他们认为这些人在解读困难语言方面受过最好的训练。随着战争的临近，为了分析类似恩尼格玛的复杂编码设备，政府编码与密码学院（GS&CS）显然也需要数学家。

1938年夏天，阿兰·图灵从普林斯顿返回到英国后被邀请到GC&CC总部做讲座。英国政府可能早在1936年就和他有了联系。1939年，GC&CC购买了一处地产，其处有一座坐落在伦敦东北50英里叫做布莱切利庄园的维多利亚时期官邸。[6]在某种意义上，布莱切利庄园是整个英格兰智慧的焦点，牛津大学和剑桥大学间的铁路线和向南通向伦敦南部的铁路线在这里交汇。

1939年9月1日，德国入侵波兰。两天后，英国向德国宣战。9月4日，阿兰·图灵到布莱切利庄园报到。最后，大约有一万人在那里进行拦截和破译秘密通信的工作。为了容纳每一个人，官邸周边建造了临时营房。图灵负责管理8号营房，致力于破译德国海军的密码。德军使用这些编码与潜水艇进行通信，这些潜水艇对驻扎在美国和英国间的大西洋护航舰队极具威胁。

早在1939年，英国政府就会见了一些波兰数学家，向他们了解恩尼格玛密码和bomb。图灵来到布莱切利庄园后不久，便开始重新设计并改进这些设备，也就是现在的bombe（法文）。1940年，第一台“图灵Bombe”（他们有时候这么叫）开始运行。它有一吨重，可以模拟30台并行运行的恩尼格玛密码机。[7]

在使用“图灵Bombe”攻击加密信息之前，有必要降低可能性范围。破译人员寻找crib，即那些在加密信息中经常出现的词汇或词组。它们可以确定第一个转子的位置。相同的信息被编码成两种形式进行传输的情况更有价值，它们被称作kisses。另一种技术是在各种宽度的厚白纸上打印多行字母，这与后来计算机里使用的打孔卡片很相像。分析员根据加密信息的字母在纸上打孔。通过把纸张重叠在一起可以比较同一天内获取的不同信息（它们都基于恩尼格玛的同一套格局）。因为这里使用的纸张来自附近名为班伯里（Banbury）的镇，因此这个过程称作banburismus技术。

到1941年中期，这些多样的技术经过改进，最终成功破译了恩尼格玛加密的通信，并大大降低了海军的损失。[8]阿兰·图灵在其中起到了至关重要的作用，当然在布莱切利庄园工作的很多人也理应为此受到称颂。

甚至在布莱切利庄园这样由数学家和古典文学学者构成的不寻常人群中，图灵依然因性格怪异而博得了一定的名声。

每年六月的第一周，图灵都会得一场严重的枯草热病，他会戴着军用防毒面具来遮蔽花粉，然后骑自行车去办公室。他的自行车有个毛病，车链每隔一定时间就会脱落。图灵不去修理它，而是数脚踏板转的圈数，然后赶在车链掉下之前下车，用手调整车链。[9]

1941年春，阿兰·图灵向琼·克拉克求婚，她是布莱切利庄园里为数不多的从事那种需动脑筋的文书工作的一位女性。琼·克拉克在被聘来做解码工作前在剑桥大学研究数学。求婚几天后，图灵向她坦白自己有同性恋倾向，[10]但是婚约还是维持了几个月，直到最后他觉得不得不取消它。

1942年11月，图灵前往华盛顿帮助协调英国和美国之间的密码破译工作。那次任务之后，他在贝尔实验室度过了第二年的前两个月，当时贝尔实验室设在纽约市西街。他在那里遇到了开辟数位采样理论的哈利·奈奎斯特（1889—1976）和克劳德·埃尔伍德·香农。香农的论文“通信的数学理论”（1948年）开创了信息论这一领域，他还引入了”比特”的概念。

图灵在贝尔实验室关注的是一台语音置乱设备，它主要用来保证大西洋上的电话通信安全。声波被分解至不同的频率范围，然后被数字化，最后通过模数加法来加密，这是一种不超过某一个特定值的加法（比如分钟值和秒值相加时，这个值是60）。在接收端，这些数字被解码，然后重新构成语音。

在奈奎斯特的研究、香农的论文以及语音加密设备中，我们可以看到后来引发将图片数字化为JPEG文件和将声音数字化为MP3文件等技术想法的最初由来。但是这些创新需要几十年才能变得成熟。另一方面，最早的数字计算机基本上只能处理数字，甚至连巴贝奇最初发明的差分机都只能用来打印无差错的对数表。在这个背景下，图灵机只能生成数字，而不能做其他的事情，比如不能执行普通函数，就一点也不令人惊讶了。

图灵即将通过使用数字加密其他形式的信息来把他的论文引向更不寻常的方向。他在论文的下一节论证了数字如何来表示机器本身，而非照片或歌曲。

的确，万物皆数字，甚至连图灵机本身都是数字的。

5. *Enumeration of computable sequences.*

A computable sequence γ is determined by a description of a machine which computes γ. Thus the sequence 001011011101111…is

determined by the table on p. 234, and, in fact, any computable sequence is capable of being described in terms of such a table.

5. 可计算序列的枚举

一个可计算序列γ是由计算γ的机器所描述。因此，序列001011011101111…是由第234页的表决定的。事实上，任何可计算序列都可以通过这样的表来描述。

那是本书第75页的例子II中的机器。

It will be useful to put these tables into a kind of standard form.

把这些表转换成一种标准形式是很用的。

事实上，图灵就是从第1节（本书第58页）描述的标准形式出发的。他指出，某种操作可以让机器打印或擦除符号，也可以把一个格移到左边或右边。在展示了一台这种形式的机器（本书第69页的例子I，也是图灵马上要提到的例子）后，图灵很快放弃了自己的规则。他允许一次操作打印多个字符，或者移动多个格。这些操作可以单独执行，因此机器表不会有几页长。现在他回到了他最初的约定上。

In the first place let us suppose that the table is given in the same form as the first table, for example, I on p. 233. That is to say, that the entry in the operations column is always of one of the forms *E*: *E*, *R*: *E*, *L*: *Pα*: *Pα*, *R*: *Pα*, *L*: *R*: *L*: or no entry at all.

首先，我们假设表仍以第233页例I中的形式给出。也就是说，操作列的条目总是下列形式之一：*E*: *E*、*R*: *E*、*L*: *Pα*: *Pα*、*R*: *Pα*、*L*: *R*: *L*，或者没有任何条目。

图灵使用冒号来分隔九种不同的可能性，这些可能性来自于三种类

型的打印（擦除、打印字符，或者两者都不是）和三种移动（向左、向右或者不移动）的结合。

The table can always be put into this form by introducing more *m*-configurations.

通过引入更多的m-格局，总是可以把表表示成这种形式。

例如，例II（本书第77页）以格局 $\mathfrak{b}$ 开始：

格局		行为	
m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$		$P\text{ə}, R, P\text{ə}, R, P0, R, R, P0, L, L$	$\mathfrak{o}$

要想遵循图灵最初的（和恢复后的）约定，这一格局必须分成六个简单的格局。对于其他格局，我将使用德语小写字母c、d、e、g和h（f在原始表中已经使用过）表示。

格局		行为	
m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$		$P\text{ə}, R$	$\mathfrak{c}$
$\mathfrak{c}$		$P\text{ə}, R$	$\mathfrak{d}$
$\mathfrak{d}$		$P0, R$	$\mathfrak{e}$
$\mathfrak{e}$		R	$\mathfrak{g}$
$\mathfrak{g}$		$P0, L$	$\mathfrak{h}$
$\mathfrak{h}$		L	$\mathfrak{o}$

现在，每个操作都只包含一个打印操作（或没有）和紧跟着的一个可能左移一格或右移一格的操作。

Now let us give numbers to the *m*-configurations, calling them q_1, …, q_R, as in §1. The initial *m*-configuration is always to be called q_1.

现在我们给这些m-格局编号，正如§1中分别把它们称作q_1，q_2，…，q_R。初始m-格局总是q_1。

如果一台机器刚好有237种不同的m-格局，则将它们标记为q_1到q_{237}。

修改例II的开头，前六个m-格局分别重命名为q_1到q_6。图灵常用的初始m-格局的符号 $\mathfrak{b}$ 变成了q_1。现在的表如下所示。

格局		行为	
m-格局	符号	操作	最终m-格局
q_1		Pə, R	q_2
q_2		Pə, R	q_3
q_3		$P0, R$	q_4
q_4		R	q_5
q_5		$P0, L$	q_6
q_6		L	q_7

We also give numbers to the symbols $S_1, \ldots, S_m$

[240]

and, in particular, blank=S_0, 0=S_1, 1=S_2.

我们也为符号S_1，…，S_m编号，特别是，空＝S_0，0＝S_1，1＝S_2。

符号下标为1表示的是0，符号下标为2表示的是1，这种方法虽然有点令人困惑，但是我们必须适应它。例II中的机器也需要打印ə和x，因此需要为这台机器定义下面的等价关系。

S_0代表空，

S_1代表0，

S_2代表1，

S_3代表ə，

S_4代表x。

计算2的平方根的机器需要从S_0到S_{14}的符号。

例II中机器的前六个格局现在如下所示。

格局		行为	
m-格局	符号	操作	最终m-格局
q_1		PS_3, R	q_2
q_2		PS_3, R	q_3
q_3		PS_1, R	q_4
q_4		R	q_5
q_5		PS_1, L	q_6
q_6		L	q_7

The lines of the table are now of form

m-config.	*Symbol*	*Operations*	*Final m-config.*	
q_i	S_j	PS_k, L	q_m	(N_1)
q_i	S_j	PS_k, R	q_m	(N_2)
q_i	S_j	PS_k	q_m	(N_3)

现在表中各行的形式如下：

m-格局	符号	操作	最终m-格局	
q_i	S_j	PS_k, L	q_m	(N_1)
q_i	S_j	PS_k, R	q_m	(N_2)
q_i	S_j	PS_k	q_m	(N_3)

规定使用统一的命名系统使得这些行都拥有非常相似的模式。图灵区分了一般情形下的三种不同的标准形式。

在最右端，图灵把这三种标准形式记作N_1、N_2和N_3。这三种形式都会打印一些字符，不同之处在于读写头是移至左边、右边或是不动。

擦除操作呢？因为图灵使用S_0来表示一个空的符号，所以擦除操作可以只通过简单地打印S_0来完成。

Lines such as

$q_i\ S_j\ E,\ R\ q_m$

are to be written as

$q_i\ S_j\ PS_0,\ R\ q_m$

如下这样的行：

$q_i\ S_j\ E,\ R\ q_m$

可以写成：

$q_i\ S_j\ PS_0,\ R\ q_m$

不打印任何符号的右移或者左移操作，可以写成重新打印扫描符：

and lines such as

$q_i\ S_j\ R\ q_m$

to be written as

$q_i\ S_j\ PS_j,\ R\ q_m$

In this way we reduce each line of the table to a line of one of the forms (N_1), (N_2), (N_3).

而如下这样的行：

$q_i\ S_j\ R\ q_m$

可以写成：

$q_i\ S_j\ PS_j, R\ q_m$

通过采用这种方法，我们可以把表的每一行都简化为（N_1）、（N_2）或（N_3）三种形式中的一种。

我一直用例II表中的第一个格局来阐明标准化表的过程，但是这个格局的符号列什么都没有，因为这个格局所做的操作与符号无关。由于一台机器以一个空白纸带开始，因此它读取的符号是一个空白。将例II中的第一个格局转换成标准形式后，结果如下：

格局	行为		
m-格局	符号	操作	最终m-格局
q_1	S_0	PS_3, R	q_2
q_2	S_0	PS_3, R	q_3
q_3	S_0	PS_1, R	q_4
q_4	S_0	PS_0, R	q_5
q_5	S_0	PS_1, L	q_6
q_6	S_0	PS_0, L	q_7

这个相当简单。我们来看看例II机器中的第二个m-格局：

o	1	R, Px, L, L, L	o
	0		q

这个m-格局 o 编号后表示成q_7。当扫描到的字符是1时，读写头向右移动一次，然后向左移动三次。这样的三次左移需要另外三个m-格

局q_8、q_9和q_{10}。m-格局将 q 变成q_{11}。m-格局q_7如下：

格局	行为		
m-格局	符号	操作	最终m-格局
q_7	S_2	PS_2, R	q_8
q_7	S_1	PS_1	q_{11}

在这两种情况下，机器都打印扫描到的字符。m-格局q_8、q_9和q_{10}如下：

q_8 S_0 PS_4, L q_9
q_9 S_2 PS_2, L q_{10}
q_{10} S_0 PS_0, L q_7

问题出自符号列。你需要知道机器接下来会遇到哪个字符，才能保证正确填写该列。对于q_8，机器扫描到一个空格，然后打印一个x。一旦它向左移动，接下来被扫描到的是哪个字符呢？它应该是刚才在q_7里扫描的那个1，但是在其他情况下，结果可能不会如此明显。”任一”、”否”或者”其他”在这个模式下都不起作用，在某些情况下，你可能不得不为机器使用的每一个单独的字符添加特定的格局。

这样的确有点乱，但是我们总是只涉及有限个字符，因此这些操作一定可以完成。假设我们已经将某台机器的所有格局都转换成了图灵记作（N_1）、（N_2）和（N_3）的标准形式。当完成转换时，我们会丢掉原始的表，这样会丢失任何信息吗？是的，确实丢失了一点信息。我们知道S_0是空，S_1是0，S_2是1，但是我们不再知道S_3、S_4等代表的准确字符是什么。这个应该没有关系。机器在内部使用这些字符。要紧的是它们必须是唯一的。我们真正关心的只是机器打印的0和1的数字串，而不关心它使用什么作为缓存记号。

我们可以不使用表，而用m-格局、符号、L和R的组合来表示每个格局。

From each line of form (N_1) let us form an expression $q_iS_jS_kLq_m$;

对于形式为（N_1）的每行，我们使用表达式$q_iS_jS_kLq_m$。

这种形式有时被称作五元组，因为它是由五个元素组成的。尽管这样表达让人感觉很神秘，但它仍然是可读的：“在m-格局q_i中，当扫描到字符S_j时，打印字符S_k，然后向左移，最后转向m-格局q_m”。对于N_2和N_3形式的写法是相似的。

from each line of form (N_2) we form an expression $q_iS_jS_kRq_m$; and from each line of form (N_3) we form an expression $q_iS_jS_kNq_m$.

形式为N_2的每行都可以表示为$q_iS_jS_kRq_m$，N_3则表示为$q_iS_jS_kNq_m$。

注意，如果读写头不移动，那么对应的字母将是N（意思是不移动）。

Let us write down all expressions so formed from the table for the machine and separate them by semi-colons. In this way we obtain a complete description of the machine.

我们把机器表中这样形成的表达式写下来，并且用分号分隔开来。如此一来，我们就得到机器的完整描述。

图灵很快会给出一个例子。每个格局都是一个五元组，一个完整的机器就被表示成一连串的五元组。（有趣的是，这些五元组不必按照某个特定的顺序排列。就像在程序语言中，每个语句以标签开始，以goto语句结束。）

下面进行的替换是彻底的替换。它去掉了所有下标，把机器转换成了一连串的大写字母。

In this description we shall replace q_i by the letter “D” followed by

the letter “*A*” repeated i times, and S_j by “*D*” followed by “*C*” repeated j times.

在下面的描述里，我们将用*D*和后面i个字母*A*来代替q_i，用*D*和后面j个字母*C*表示S_j。

比如，q_1 可以由*DA*代替，q_5 由*DAAAAA*代替（记住，第一个格局是q_1，没有q_0）。对于符号，S_0（空）表示成*D*，S_1（符号0）表示成*DC*，S_2（符号1）表示成*DCC*，S_3表示成*DCCC*，以此类推。

This new description of the machine may be called the *standard description* (S.D). It is made up entirely from the letters *“A”, “C”, “D”, “L”, “R”, “N”*, and from “;”.

这种新的机器描述方法称为标准描述（S.D）。它完全是由字母*A*、*C*、*D*、*L*、*R*、*N*和分号“；”组成的。

其中，*L*、*R*和*N*表示读写头的移动。分号用来分隔每个格局。

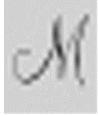
If finally we replace *“A”* by “1”, *“C”* by “2”, *“D”* by “3”, *“L”* by “4”, *“R”* by “5”, *“N”* by “6”, and “;” by “7” we shall have a description of the machine in the form of an arabic numeral.

如果最后用1代替*A*，2代替*C*，3代替*D*，4代替*L*，5代替*R*，6代替*N*，7代替”；”，我们将得到阿拉伯数字形式的机器描述。

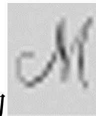
这是很重要的一步。图灵已经把他的机器标准化到了可以用一个整数来唯一确定一台机器，这一整数将机器的所有状态进行了编码。哥德尔曾在他的不完备性定理中将每个数学表达式转化成唯一的数字，图灵无疑在此受到了哥德尔方法的启发。

The integer represented by this numeral may be called a *description number* (D.N) of the machine. The D.N determine the S.D and the structure of the

[241]

machine uniquely. The machine whose D.N is *n* may be described as (n).

由这些数字表示的整数可以称作机器的描述数（D.N）。D.N唯一决定了S.D以及机器的结构。

D.N为*n*的机器可以描述为(*n*)。

图灵又引入了另一种字体。他将使用这种手写字体来表示整台机器。

To each computable sequence there corresponds at least one description number, while to no description number does there correspond more than one computable sequence.

每个可计算序列至少对应一个描述数，但不存在一个描述数对应多个可计算序列。

由于五元组的序列无关紧要，因而打乱这些五元组的顺序，不会影响机器计算出的序列。很明显，可以有多个描述数与一个可计算序列相关，但是每个描述数都定义了一台只能产生一个可计算序列（至少在以一个空白纸带开始的情况下）的机器。

不知不觉地，图灵就得到了他在论文开始提到的一个结果：

The computable sequences and numbers are therefore enumerable.

因此，可计算序列和可计算数是可数的。

你可以通过列出所有可能的描述数来枚举可计算序列，因为描述数其实就是整数。图灵没有陈述的一点是，可计算数只是实数一个可数的子集。因为可计算数是可数的，而实数是不可数的，所以存在很多不可计算的实数。这一主题将在后面更多地探讨。

Let us find a description number for the machine I of §3.

我们来寻找§3中机器I的描述数。

最初，那台机器是由如下表定义：

格局		行为	
m-格局	符号	操作	最终 m-格局
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$

When we rename the *m*-configurations its table becomes:

$q_1\, S_0\, PS_1, R\, q_2$
$q_2\, S_0\, PS_0, R\, q_3$
$q_3\, S_0\, PS_2, R\, q_4$
$q_4\, S_0\, PS_0, R\, q_1$

重命名这些m-格局之后，表变成：

$q_1\, S_0\, PS_1, R\, q_2$

$q_2\ S_0\ PS_0,R\ q_3$
$q_3\ S_0\ PS_2,R\ q_4$
$q_4\ S_0\ PS_0,R\ q_1$

这是一种很直接的转换。

Other tables could be obtained by adding irrelevant lines such as

$q_1\ S_1\ PS_1,R\ q_2$

通过增加如下的无关行可以得到其他表：

$q_1\ S_1\ PS_1,R\ q_2$

也就是说，增加一些不起作用的行可以得到生成相同可计算序列的其他表。如果机器开始时纸带是空白的，并且每当打印一格后它总是向右移动，那么机器将永远扫描不到数字0。

Our first standard form would be

$$q_1\ S_0\ S_1\ R\ q_2;\ q_2\ S_0\ S_0\ R\ q_3;\ q_3\ S_0\ S_2\ R\ q_4;\ q_4\ S_0\ S_0\ R\ q_1;.$$

我们得到的第一个标准形式如下：

$$q_1S_0S_1Rq_2;\ q_2S_0S_0Rq_3;\ q_3S_0S_2Rq_4;\ q_4S_0S_0Rq_1;.$$

这一标准形式是通过取表中的四行，并用分号将这四行代表的格局分开得到的。把它转化成标准描述，要求将q_i替换成D与i（1个或更多）个A相连的序列，将S_j替换成D与j（0个或更多）个C相连的序列。

The standard description is
DADDCRDAA;DAADDRDAAA;

DAAADDCCRDAAAA;DAAAADDRDA;

标准描述为：
DADDCRDAA;DAADDRDAAA;
DAAADDCCRDAAAA;DAAAADDRDA;

标准描述很难阅读，但是经常使用，因此你应该努力去熟悉它。为了把它解码成其原来的组成部分，我们先来关注每一个*D*。每个*D*都代表了一个格局或者一个符号。

〉 如果这个*D*后紧跟若干个*A*，那么它代表了一个格局。这个格局编号是*A*的数目。

〉 如果*D*后不是*A*，那么它是一个符号。这种情况下，*D*后跟随的是0个或更多个*C*。如果只是*D*本身，那么它表示一个空格；否则，*DC*表示0，*DCC*表示1，与更多的*C*相连表示其他符号。

图灵更多使用的是标准描述，而不是描述数。描述数更多地存在于抽象意义上，图灵不对这些数字进行任何计算。在他所展示的例子中，你可以用1代替*A*，用2代替*C*，用3代替*D*，用5代替*R*，用7代替分号，来得到一个描述数。

A description number is
313325311731133531117311133225311117311113353 17
and so is
3133253117311335311173111332253111173111133531731323253

一个描述数是：
31332531173113353111731113322531111731111335317， 而
3133253117311335311173111332253111173111133531731323253117
也是一个描述数。

第二个数除了尾部增加了一些数字（31323253117）之外，和第一个数是一样的，这些增加的数字对应了图灵定义的”不相关的”格局$q_1S_1S_1Rq_2$。问题在于：这两个数定义了两台不同的机器，而这两台机器又计算得到两个完全相同的数。回想一下，这个数是1/3的二进制形

式。也就是说，一台机器的格局重组后仍然可以计算产生相同的数，但是它们的描述数是不同的。

这个数太大了！显然，图灵并不在意这些数有多大。例如，为了表示出q_{35}，他或许能想出一种办法把35放进描述数中，但是他没有这么做。为表示q_{35}，标准描述要使用如下序列：

DAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

描述数包括如下数位：

31111111111111111111111111111111111

并且这些数位不止被包含一次，而是至少两次。

这种方法带来的好处很有意思。想想一台计算π的图灵机。正常情况下，我们用一个无穷序列来表示π的数位：

π=3.14159265358979323846264338327 95…

现在，我们可以使用一个有限的整数来表示π，这个整数就是计算π的数位的图灵机的描述数。哪种π的表示更好呢？是前32个数位再跟上省略号？还是有多长耐心就能生成多少位数字的图灵机描述数？在某种意义上，描述数是π的一种更基本的数字表示方式，因为它描述了计算这个数的算法。

事实上，把每台机器归纳为一个数，图灵已经使仅通过枚举正整数便产生机器变为可能了。不是每一个正整数都是一个有效的图灵机描述数，而且很多有效的描述数也不能描述非循环机，但这种枚举必然包含了所有的非循环图灵机，其中每台这样的机器都对应着一个可计算数。因此，可计算数是可枚举的。

这是一个很重要的发现，尽管它也可能是令人不安的，因为它意味着，大部分的——或者根据我们已有的关于实数范围的知识，几乎全部的——实数都不是可计算的。

这个出乎意料的发现，连同一些数学悖论和量子引力的研究，促使数学家格雷戈里·蔡廷提出疑问：”实数究竟有多真实？”[11]能证明实数确实存在的证据的确很少。

现在的程序员很自然地就能想到可以表示成数的计算机程序，因为一个程序的可执行文件只是一些连续的字节。通常我们不会把这些字节当作简单的数，但其实是可以的。例如，MS Word 2003的可执行文件是WinWord.exe，其大小为12 047 560字节。它大约包含9600万位，或者2900万十进制位，因此表示WinWord.exe的数接近$10^{29\,000\,000}$。这的确是一个很大的数。如果在一本每页有50行的书里，按每行打印50位写下这个数，那么它要占用11 000多页。这是一个比著名的googol（即10的100次方）更加巨大的数，但是它仍然是一个有限数。WinWord.exe只是在枚举整数的过程中出现的众多可执行程序（好比所有可能的图灵机）之

一，同时我们还会枚举出其他所有的文字处理软件，甚至是那些还没编写好的。

以备将来使用，图灵在本节末给出一个定义。

> A number which is a description number of a circle-free machine will be called a *satisfactory* number. In §8 it is shown that there can be no general process for determining whether a given number is satisfactory or not.

> 非循环机的描述数称作可接受数。在§8中，我们会指出，不存在一个通用程序来判定某个给定数是否是可接受数。

很容易判断某个特定的整数是否是一个明确定义的描述数，但是图灵坚持认为，不存在一个通用过程来判断某个特定的描述数是否可以表示一个非循环机，并且可以按要求打印一连串的0和1。没有一个通用过程可以用来确定一个机器是否可能扫描到它不希望遇到的字符，或者在打印空格时陷入无限循环中，以及判断机器是否崩溃、发热、垮掉或者数位丢失得无影无踪。

---

[1] 安德鲁·霍奇斯，*Alan Turing: The Enigma*（Simon & Schuster，1983），138。

[2] David Kahn，*Seizing the Enigma: The Race to Break the German U-Boat Codes*，*1939-1943*（Houghton-Mifflin，1991），ch. 3。

[3] 马里安·雷耶夫斯基，“How Polish Mathematicians Deciphered the Enigma”，*Annals of the History of Computing*，Vol. 3，No. 3（July 1981），213-234。也可参见 Elisabeth Rakus-Andersson，“The Polish Brains Behind the Breaking of the Enigma Code Before and During the Second World War” 见于Christof Teuscher，ed.，*Alan Turing: Life and Legacy of a Great Thinker*（Springer，2004），419-439。

[4] 雷耶夫斯基，“How Polish Mathematicians Deciphered the Enigma”，226。

[5] Kahn，*Seizing the Enigma*，73。

[6] 霍奇斯，*Alan Turing*，148。

[7] Stephen Budiansky，*Battle of Wits: The Complete Story of Codebreaking in World War II*（Free Press，2000），155 。同样可见于

Jack Gray and Keith Thrower，*How the Turing Bombe Smashed the Enigma Code*（Speedwell，2001）。
[8] 霍奇斯，*Alan Turing*，218-9。
[9] I. J. Good，“Early Work on Computers at Bletchley”，*Annals of the History of Computing*，Vol. 1，No. 1（July 1979），41。
[10] 霍奇斯，*Alan Turing*，206。
[11] 格雷戈里·蔡廷，“How Real are Real Numbers?”，*International Journal of Bifurcation Chaos*，Vol. 16（2006），1841-1848。重印于Gregory J Chaitin，Thinking About Gödel Turing: Essays on Complexity，1970-2007（World Scientific，2007），267-280。

第9章
通　用　机

图灵在论文下一节所描述的机器就是我们今天熟知的通用图灵机，之所以称为通用机，是因为我们只要有了这种机器就足够了。之前介绍的特殊计算机器既不能保证按类似的方式执行，甚至也没有通用的零件。而通用机在获得其他机器的标准描述后就可以模拟它们。如今我们可以说，通用机是可编程的。

6. *The universal computing machine.*

It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathcal{U}$ is supplied with a tape on the beginning of which is written the S.D of some computing machine $\mathcal{M}$,

[242]

then $\mathcal{U}$ will compute the same sequence as $\mathcal{M}$. In this section I explain in outline the behaviour of the machine. The next section is devoted to giving the complete table for $\mathcal{U}$.

6. 通用计算机器

发明一台可以计算任何可计算序列的机器是可能的。如果为机

器$\mathcal{U}$提供一条纸带，纸带开头写入的是某台计算机器$\mathcal{M}$的标准描述，那么$\mathcal{U}$可以计算出与$\mathcal{M}$相同的序列。在这一节中，我将概括地介绍这种机器的行为。下一节着重给出$\mathcal{U}$的完整表。

这里再次使用了手写字体。图灵使用$\mathcal{M}$表示任意一台机器，而使用$\mathcal{U}$表示通用机。

当我们说到计算机程序的时候，通常会涉及输入和输出。一段程序读取输入，写出输出。但是到目前为止，我们描述的机器基本上都没有输入，因为它们是以空白纸带开始的。这些机器产生0和1序列形式的输出，输出序列中可能会临时散布一些其他用作标记或便笺的字符。

相反，通用机$\mathcal{U}$要求实际输入，特别地，要求纸带上包含$\mathcal{M}$的标准描述&md;&md;字母*A*、*C*、*D*、*L*、*N*、*R*构成的描述了$\mathcal{M}$所有格局的序列。机器$\mathcal{U}$读入并解释这个标准描述，然后打印出和$\mathcal{M}$一样的输出。

但是这并不完全正确：$\mathcal{U}$的输出和$\mathcal{M}$的输出可能不完全相同。通常情况下，$\mathcal{U}$不能完全模仿$\mathcal{M}$。机器$\mathcal{M}$可能以一个空白的纸带开始，但$\mathcal{U}$不是这样，提供给它的纸带已经包含了$\mathcal{M}$的标

准描述。如果$\mathcal{M}$违反了图灵的约束，向两个方向输出，会出现什么情况呢？准确地模仿$\mathcal{M}$将导致重写$\mathcal{M}$的标准描述。

图灵说，$\mathcal{U}$会配有一条以机器$\mathcal{M}$的标准描述开头的纸带。如果纸带的两个方向都是无限延伸的，就没有“开头”一说了。图灵暗含的意思是，将$\mathcal{U}$的输出写到纸带上的标准描述之后。

如果考虑机器只向一个方向打印（正是图灵的约束），那么我们能写出一个通用机，让它读取纸带开头处某台机器的标准描述，然后在标准描述后面无限的空白区域里精确地复制出该机器的输出吗？

这似乎不太可能。无疑，通用机需要自己的便笺空间，因此它的输出和它尝试模仿的机器的输出是不同的。即使我们仅仅要求通用机复制机器$\mathcal{M}$的F-格，这台通用机也明显会比图灵之前描述的机器复杂得多。

图灵不能保证他的通用机能够如实地复制它正在模仿的机器的输出。他只说：“$\mathcal{U}$会计算出与$\mathcal{M}$相同的序列。”事实上，$\mathcal{U}$会在序列之外打印很多额外的输出。

图灵从一个很奇怪的角度着手通用机的设计。

> Let us first suppose that we have a machine $\mathcal{M}'$ which will write down on the *F*-squares the successive complete configurations of $\mathcal{M}$.

首先假设我们有一台机器$\mathcal{M}$′，它会在纸带的F-格处写下$\mathcal{M}$的连续完全格局。

如前所述，完全格局是完成一个操作后纸带的快照，连同读写头的位置和下一个m-格局。这些连续的完全格局提供了机器完整的历史操作记录。

These might be expressed in the same form as on p. 235, using the second description, (C), with all symbols on one line.

这些可以使用同第235页一样的形式表达，使用第二种描述(C)，并且所有的符号都在同一行。

本书第82页用了如下的序列展示信息：

b :əəo0　　0:əəq0　　0:...

在该记法中，连续的完全格局间使用冒号隔开。在每一个完全格局中，表示下一个m-格局的德文字母放在下一个扫描符前面。

Or, better, we could transform this description (as in §5) by replacing each *m*-configuration by *"D"* followed by *"A"* repeated the appropriate number of times, and by replacing each symbol by *"D"* followed by *"C"* repeated the appropriate number of times. The numbers of letters *"A"* and *"C"* are to agree with the numbers chosen in §5, so that, in particular, "0" is replaced by *"DC"*, "1" by *"DCC"*, and the blanks by *"D"*.

或者，更好的方法是，用*D*后跟适当数目的*A*所构成的字符串代替每个m-格局，用*D*后跟适当数目的*C*所构成的字符串代替每个符号，以此来转换当前的描述（如§5）。字母*A*和*C*的数目和§5中

的数目是一致的，因此，0使用*DC*代替，1用*DCC*代替，空则由*D*代替。

图灵提出这种标准描述（他这么称）来编码机器的状态。现在，他提出再使用它来表示完全格局。

> These substitutions are to be made after th complete configurations have been put together, as in (C). Difficulties arise if we do the substitution first.

> 只有在完全格局组合在一起后，才能进行这些替换，比如在(C)中。如果先进行替换会出现问题。

我猜想，图灵这里想要表达的意思是，由于m-格局和符号要被多个符号代替（比如，1变成*DCC*），因此在查找下一个格局的时候必须注意滑动一些格，以防破坏某个符号的编码。

> In each complete configuration the blans would all have to be replaced by *“D”*, so that the complete configuration would not be expressed as a finite sequence of symbols.

> 每个完全格局中的空白都必须用*D*替换，因此，完全格局不能表示成一个有限的符号序列。

字母*D*代表一个空格。图灵不想在一个完全格局中出现任何中断。他想使每个完全格局都是一个没有中断、连续的字符串。他的原话是这样的：“因此，完全格局不能表示成一个有限的字母序列。”这样的表达不是很明确。我觉得“不能”应该是“现在”。很明显，他不想用一个由符号*D*构成的无限序列来表示一个空白的纸带，每一个完全格局都是有限的。

If in the description of the machine II of §3 we replace “ ” by *“DAA”*, “ ” by *“DCCC”*, “ ” by *“DAAA”*, then the sequence (C) becomes:

DA: *DCCCDCCCDAADCDDC*: *DCCCDCCCDAAADCDDC*:… (C_1)

(This is the sequence of symbols of *F*-squares.)

如果在§3机器II的描述中，把 替换成*DAA*，把 替换成*DCCC*，把 替换成*DAAA*，则序列(C)变成：

DA：*DCCCDCCCDAADCDDC*：*DCCCDCCCDAAADCDDC*：…(C_1)

（这是F-格处的符号序列。）

图灵并没有提到他所做的所有替换。他还把 替换成了*DA*，空白替换成了*D*，0替换成了*DC*。

括号里的说明指的是图灵提出的机器$\mathcal{M}$'的输出方式。正常的机器$\mathcal{M}$在F-格处打印0和1，并且用E-格处的其他符号来辅助计算0和1。机器$\mathcal{M}$'在F-格处打印机器$\mathcal{M}$的连续的完全格局，并且使用E-格辅助构造这些连续的完全格局。

用这种方式表示的完全格局是很难阅读的。正如前面所说的，需要注意的是*D*，它可以表示一个格局或者一个符号。

〉 如果*D*后接的是一个或多个*A*,那么它是一个格局。格局编号是A的数

目。

〉 如果*D*后接的不是*A*，则它是一个符号。在这种情况下，*D*后接的可能是0个或多个*C*。如果*D*单独出现，则它代表一个空白；否则，*DC* 代表0，*DCC* 代表1，后接多个*C*则指示其他符号。

It is not difficult to see that if $\mathcal{M}$ can be constructed, then so can $\mathcal{M}$′. The manner of operation of $\mathcal{M}$′ could be made to depend on having the rules of operation (*i.e.*, the S.D) of $\mathcal{M}$ written somewhere within itself (*i.e.* within $\mathcal{M}$′); each step could be carried out by referring to these rules.

不难看出，如果我们能构造出$\mathcal{M}$，那么我们也能构造出$\mathcal{M}$′。$\mathcal{M}$的操作规则（即标准描述）可以写进自身（即$\mathcal{M}$′）的某个地方由它来决定$\mathcal{M}$′的操作方式，每一步都可以由这些规则算出来。它的每一步操作都可以参考这些规则来执行。

把$\mathcal{M}$的标准描述写进$\mathcal{M}$′的某个地方，这是一个全新的概念。

它是如何写的呢？如何访问它呢？图灵追求机器$\mathcal{M}$′的方式有点偏离

他的目标，虽然看上去$\mathcal{M}$'确定是可以构造出来的。

> We have only to regard the rules as being capable of being taken out and exchanged for others and we have something very akin to the universal machine.

> 我们只需要认为这些规则是可以取出并替换的，那么我们便得到了某种本质上非常类似通用机的东西。

现在问题变得清晰点儿了。图灵在本节开头说过，要为$\mathcal{U}$提供一个包含$\mathcal{M}$的标准描述的纸带。这正是“这些规则可以取出并替换的”要表达的意思。我们可以为$\mathcal{U}$提供一个包含任何想要$\mathcal{U}$模仿的机器的标准描述的纸带。

抽象地来看，$\mathcal{U}$现在变得几乎是，嗯，也不完全是显然的，但仍然已经简单了很多。$\mathcal{U}$以一个打印着$\mathcal{M}$的标准描述的纸带开始。这个标准描述负责打印$\mathcal{M}$的连续完全格局。标准描述和完全格局使用相同的编码方式：每个完全格局包含一个字母序列，这些序列大多指示纸带上打印的符号。每个完全格局还包含了一个以*D*开头、后接若干个*A*的字符串，用来表示位于扫描符前面的下一个格局。例如：

DAAADCC

完全格局中的这个字母序列表示下一个m-格局是q_3，下一个扫描符

是1。$\mathcal{M}$的标准描述的某个位置会出现与之完全匹配的字母序列。（如果没有，一定是有地方出错了，从而$\mathcal{M}$不是非循环的。）$\mathcal{U}$要确定下一个格局，所需要做的就是寻找一个匹配。一旦$\mathcal{U}$找到了匹配的格局，它可以直接访问这个格局的操作——要打印的符号、指示如何移动读写头的编码以及下一个m-格局。$\mathcal{U}$必须基于最后的完全格局创建一个新的完全格局，并且把要打印的字符和下一个m-格局合并进来。

一旦考虑到机器操作的第一个完全格局是平凡的，并且从一个完全格局转向下一个完全格局的步骤只有很小的变化，那么通用机可能就简单多了。事实上，这只是一个比较和复制符号的问题，图灵已经定义了一组执行这些常用操作的m-函数。

至此，图灵仍然在谈论$\mathcal{M}'$，而不是$\mathcal{U}$，而且$\mathcal{M}'$只打印$\mathcal{M}$的完全格局。

One thing is lacking: at present the machine $\mathcal{M}'$ prints no figures.

我们忽略了一个问题：现在的机器$\mathcal{M}'$没有打印数字。

是的。激动之余，我们忘记了$\mathcal{M}'$（或者$\mathcal{U}$）都只在F-格处打

印了$\mathcal{M}$的连续完全格局，这些格局由字母*A*、*C*、*D*和冒号分隔符构成，它也可能使用E-格作为一个便笺。我们的实际目的是打印0和1。

We may correct this by printing between each successive pair of complete configurations the figures which appear in the new configuration but not in the old. Then (C_1) becomes

DDA : 0 : 0 : *DCCCDCCCDAADCDDC* : *DCCC*... (C_2)

It is not altogether obvious that the *E*-squares leave enough room for the necessary “rough work”, but this is, in fact, the case.

我们可以在每个连续的完全格局对间打印数字，这些数字在新格局里出现而未在旧格局里出现，以此来修正这个问题。(C_1)变成：

DDA : 0 : 0 : *DCCCDCCCDAADCDDC* : *DCCC*... (C_2)

总体而言，E-格可以为这些必要的”粗重工作”留下足够的空间，这一点虽然不明显，但事实上确实是这样的。

(C_2)开头多了一个*D*，这是印刷错误。(C_2)和(C_1)开头的区别仅在于两个0以及之间的冒号。它们是第一步操作的结果，因此打印在了第一个完全格局的后面。

图灵想要$\mathcal{M}$'（和$\mathcal{U}$）打印和$\mathcal{M}$一样的0和1，如此就可以说$\mathcal{M}$'和$\mathcal{M}$计算出相同的序列。区别仅是，这些数位被埋藏在输出里，夹在了机器的一个又一个完全格局之间。

这就是为什么图灵要求他的机器连续打印计算得到的数，并且数一旦被打印就不能再改变。如果没有这个要求，$\mathcal{M}$'（和$\mathcal{U}$）打印出的数可能会相当混乱。

图灵说$\mathcal{M}'$应该打印“在新格局里出现，而没有在旧格局里出现的”全部的数字（0或1）。当把机器简化到标准形式（也就是，每次操作只包含一个要打印的符号，且读写头移动一次）时，机器在去其他地方的路上会经常扫描到一个符号0或1。在这些情况下，机器必须重新打印0或1。$\mathcal{M}'$应该忽略$\mathcal{M}$在其自身上打印0或1的次数。$\mathcal{M}'$（也暗指通用机）应该只在扫描符为空时打印0或1。

图灵总结本节时，提议完全格局应该表示成数字形式，尽管他从来没有使用过这种方式。

> The sequences of letters between the colons in expressions such as (C_1) may be used as standard descriptions of the complete configurations. When the letters are replaced by figures, as in §5, we shall have a numerical
>
> [243]
>
> description of the complete configuration, which may be called its description number.

可以把表达式（如C_1）中冒号之间的字母序列作为完全格局的标准描述。如果用数字代替这些字母，就像在§5中那样，那么我们可以得到这个完全格局的数字表示，也可以称作它的描述数。

现在，让我们暂时忘记$\mathcal{M}'$，开始考虑$\mathcal{U}$。

现在大家已经知道，图灵对于通用机的描述中有少量错误。（令人吃惊的是，里面包含的错误如此之少，尤其是考虑到图灵不能在真实的计算机上模拟它。）在分析通用机的过程中，我很感激埃米尔·波斯特的纠正[1]和唐纳德·戴维斯所做的分析。[2]

通用机对图灵论文其余部分的论点很重要，事实上，他完整地构造了这样一台机器，证明这样的机器是存在的，整个过程极其详细。然而，一旦你理解了它的基本机制，就会发现这些细节太过冗余。因而即便你没有消化图灵描述中的每一个符号和函数，也没有关系。

7. *Detailed description of the universal machine.*

A table is given below of the behaviour of this universal machine. The *m*-configurations of which the machine is capable are all those occurring in the first and last columns of the table, together with all those which occur when we write out the unabbreviated tables of those which appear in the table in the form of *m*-functions. *E.g.*, $\mathfrak{e}(\mathfrak{anf})$ appears in the table and is an *m*-function.

7. 通用机的详细描述

下表给出了通用机的行为。机器能执行的m-格局就是表中第一列和最后一列中的全部格局，以及把表中的m-函数展开成未缩略表后将会出现的所有格局。比如，表中出现的$\mathfrak{e}(\mathfrak{anf})$就是一个m-函数。

m-格局$\mathfrak{anf}$是图灵通用机的一部分。在机器的描述即将结束时，某个格局在其最终m-格局中包含了$\mathfrak{e}(\mathfrak{anf})$。$\mathfrak{e}$的骨架表在图灵论文的第239页（本书第113页）。

$\mathfrak{e}(\mathfrak{C})$	ə	R	$\mathfrak{e}_1(\mathfrak{C})$
	Not ə	L	$\mathfrak{e}(\mathfrak{C})$
$\mathfrak{e}_1(\mathfrak{C})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{C})$
	None		$\mathfrak{C}$

在使用$\mathfrak{C}$代替$\mathfrak{anf}$后，图灵展示了未经缩写的表。

Its unabbreviated table is (see p. 239)

$\mathfrak{e}(\mathfrak{anf})$	ə	R	$\mathfrak{e}_1(\mathfrak{anf})$
	not ə	L	$\mathfrak{e}(\mathfrak{anf})$
$\mathfrak{e}_1(\mathfrak{anf})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{anf})$
	None		$\mathfrak{anf}$

Consequently $\mathfrak{e}_1(\mathfrak{anf})$ is an *m*-configuration of $\mathcal{U}$.

其未缩略表如下（见第239页）：

$\mathfrak{e}(\mathfrak{anf})$	ə	R	$\mathfrak{e}_1(\mathfrak{anf})$
	not ə	L	$\mathfrak{e}(\mathfrak{anf})$
$\mathfrak{e}_1(\mathfrak{anf})$	Any	R, E, R	$\mathfrak{e}_1(\mathfrak{anf})$
	None		$\mathfrak{anf}$

因此，$\mathfrak{e}_1(\mathfrak{anf})$是$\mathcal{U}$的一个m-格局。

图灵从描述一个包含某台机器的标准描述的纸带开始谈起。这就是通用机将要读取并解释的纸带。

When $\mathcal{U}$ is ready to start work the tape running through it bears on it the symbol ə on an *F*-square and again ə on the next *E*-square; after this, on *F*-squares only, comes the S.D of the machine followed by a double colon “::” (a single symbol, on an *F*-square). The S.D consists of a number of instructions, separated by semi-colons.

当$\mathcal{U}$准备开始工作的时候，穿过它的纸带会在某个F-格上印有符号ə，下一个E-格上又是一个ə。在此之后是只打在F-格上的机器标准描述，然后是一个双冒号“：：”（单个符号，出现在F-格处）。标准描述由若干指令构成，中间用分号隔开。

顺便指出，这是图灵第一次在论文中使用“指令”这个词。它用在这里很恰当，因为机器的格局在这里起了不同的作用，它们变成了通用机的指令。

在此之前（本书第127页的论文第5节中），图灵展示了每个格局后跟一个分号，然而，通用机要求每个指令以一个分号开始。这只是通用机描述里极少几个“错误”中的一个。

为了描述$\mathcal{U}$的运行过程，我们为它提供一个简单的机器$\mathcal{M}$。它是交替打印0和1的机器的一种简化形式：

m-格局	符号	操作	最终m-格局
q_1	S_0	PS_1, R	q_2
q_2	S_0	PS_2, R	q_1

这个简化的机器只包含两种格局，而不是四种，并且不跳过任何一个格。下面准备的纸带与图灵的说明相符，不过每个指令前都有一个分号。因为整个纸带很长，我在这里把它们写成两行：

ə	ə	;		D		A		D		D		C		R		D		A		A	

;		D		A		A		D		D		C		C		R		D		A		::		•••

双冒号把$\mathcal{M}$的指令与$\mathcal{U}$打印出的$\mathcal{M}$的连续完全格局分隔开。图灵提醒了我们这些指令是如何被编码的。

Each instruction consists of five consecutive parts

(i) *“D”* followed by a sequence of letters *“A”*. This describes the relevant *m*-configuration.

每条指令由五个连续的部分组成：

(i) *D*后接若干个*A*构成的序列，描述了相关的m-格局；

*D*后必须至少有一个*A*，来表示一个m-格局。也就是说，第一个格局是q_1，而不是q_0。

(ii) *“D”* followed by a sequence of letters *“C”*. This describes the scanned symbol.

(ii) *D*后接若干个*C*构成的序列，描述了扫描符；

对于符号，单独一个*D*表示一个空白；一个*D*后接一个*C*表示0；一个*D*后接两个*C*表示1。

(iii) *“D”* followed by another sequence of letters *“C”*. This describes the symbol into which the scanned symbol is to be changed.

(iv) *“L”*, *“R”*, or *“N”*, describing whether the machine is to move to left, right, or not at all.

(v) *“D”* followed by a sequence of letters *“A”*. This describes the final *m*-configuration.

(iii) *D*后接另一若干个*C*构成的序列，描述了扫描符将要转变成的符号；

(iv) *L*、*R*或*N*，描述了读写头左移、右移还是不移动；

(v) 另一个*D*后接若干个*A*构成的序列，描述了最终m-格局。

通用机需要打印完全格局，而完全格局是由字母*A*、*C*和*D*构成的，同时它还需要打印由0和1组成的可计算序列。通用机使用小写字母作为E-格中的标记。

The machine $\mathcal{U}$ is to be capable of printing *“A”*, *“C”*, *“D”*, *“0”*, “1”, *“u”*, *“v”*, *“w”*, *“x”*, *“y”*, *“z”*.

现在的机器$\mathfrak{U}$可打印*A*、*C*、*D*、0、1、*u*、*v*、*w*、*x*、*y*、*z*。

图灵忘了分号“;”（用来分隔连续的完全格局）。

The S.D is formed from “;”, “*A*”, “*C*”, “*D*”, “*L*”, “*R*”, “*N*”.

S.D由;、*A*、*C*、*D*、*L*、*R*、*N*构成。

接下来，图灵展示了通用机要求的最后一个函数。

[244]

Subsidiary skeleton table.

$\mathrm{con}(\mathfrak{C}, \alpha)$	Not A	R, R	$\mathrm{con}(\mathfrak{C}, \alpha)$	$\mathrm{con}(\mathfrak{C}, \alpha)$. Starting from an F-square, S say, the sequence C of symbols describing a configuration closest on the right of S is marked out with letters α. $\rightarrow \mathfrak{C}$.
	A	$L, P\alpha, R$	$\mathrm{con}_1(\mathfrak{C}, \alpha)$	
$\mathrm{con}_1(\mathfrak{C}, \alpha)$	A	$R, P\alpha, R$	$\mathrm{con}_1(\mathfrak{C}, \alpha)$	
	D	$R, P\alpha, R$	$\mathrm{con}_2(\mathfrak{C}, \alpha)$	
$\mathrm{con}_2(\mathfrak{C}, \alpha)$	C	$R, P\alpha, R$	$\mathrm{con}_2(\mathfrak{C}, \alpha)$	$\mathrm{con}(\mathfrak{C}, \ )$. In the final configuration the machine is scanning the square which is four squares to the right of the last square of C. C is left unmarked.
	Not C	R, R	$\mathfrak{C}$	

辅助的骨架表如下：

$\mathrm{con}(\mathfrak{C}, \alpha)$	Not A	R, R	$\mathrm{con}(\mathfrak{C}, \alpha)$	$\mathrm{con}(\mathfrak{C}, \ \alpha)$，以一个F-格（比如$S$）开始，用$\alpha$标记描述距离$S$右边最近的那个格局的符号序列$C$。然后→$\mathfrak{C}$。
	A	$L, P\alpha, R$	$\mathrm{con}_1(\mathfrak{C}, \alpha)$	
$\mathrm{con}_1(\mathfrak{C}, \alpha)$	A	$R, P\alpha, R$	$\mathrm{con}_1(\mathfrak{C}, \alpha)$	
	D	$R, P\alpha, R$	$\mathrm{con}_2(\mathfrak{C}, \alpha)$	$\mathrm{con}(\mathfrak{C}, \ )$，进入最终格局时，机器会扫描$C$的最后一格右侧的第四格，$C$没有被标记。
$\mathrm{con}_2(\mathfrak{C}, \alpha)$	C	$R, P\alpha, R$	$\mathrm{con}_2(\mathfrak{C}, \alpha)$	
	Not C	R, R	$\mathfrak{C}$	

m-函数con代表了configuration（格局），这里漏了一行[3]：

$$\mathrm{con}_1(\mathfrak{C}, \alpha) \quad \text{None} \quad PD,\ R,\ P\alpha,\ R,\ R,\ R \quad \mathfrak{C}$$

不久，我们可以看到漏掉的这一行所起的作用。

con函数的功能是用第二个参数给出的符号标记一个格局。假设读写头处在一条指令前端的分号位置上。

ə	ə	;		D		A		D		D		C		R		D		A		A	

;		D		A		A		D		D		C		C		R		D		A		::		…

con函数每次向右移动两个格，直到遇到一个字母A。然后在A的左边打印一个字符α。函数con_1继续在每个A的右边打印标记，直到遇到一个字母D。然后在这个D的右边打印一个标记，再转向con_2。函数con_2在每个C（如果C存在）的右侧打印标记。在本例中，格局中没有C，因为扫描格是一个空格，因此结果如下：

ə	ə	;		D		A		D		D		C		R		D		A		A	

;		D	α	A	α	A	α	D	α	D		**C**		C		R		D		A		::		…

骨架表中对con的解释部分有一点令人迷惑，因为图灵使用字母*C*代表定义一个格局的整个符号序列，然后又在标准描述中使用相同的字母。第二段（以“最终格局中”开头）的第一句表明读写头最后将留在格局中最后一格（也就是，扫描字符的最后一个格）右边的四个格处。“*C*未被标记”意思是“这个格局未被标记”，仅当con的第二个参数为空时才适用。

对通用机的描述仅占图灵论文中的两页。图灵前面定义的m-函数很有技巧，使得在很多情况下，$\mathfrak{U}$的m-格局只是简单地表示一个特定的函数。照例，机器仍以m-格局$\mathfrak{b}$开始。

The table for $\mathfrak{U}$.

$\mathfrak{b}$		$\mathfrak{f}(\mathfrak{b}_1, \mathfrak{b}_1, ::)$	$\mathfrak{b}$. The machine prints $:DA$ on the *F*-squares after $::$ → $\mathfrak{anf}$.
$\mathfrak{b}_1$	$R, R, P:, R, R, PD, R, R, PA$	$\mathfrak{anf}$	

机器$\mathfrak{U}$的表如下：

$\mathfrak{b}$		$\mathfrak{f}(\mathfrak{b}_1, \mathfrak{b}_1, ::)$	$\mathfrak{b}$，在$::$之后的F-格处打印$:DA$，然后→$\mathfrak{anf}$。
$\mathfrak{b}_1$	$R, R, P:, R, R, PD, R, R, PA$	$\mathfrak{anf}$	

m-函数$\mathfrak{f}$寻找把指令和完全格局隔开的双冒号。正如前面看到的，每一个完全格局展示了纸带上的所有符号，并且m-格局位于被扫描格前。当机器启动时，第一个m-格局是q_1，它的标准描述是*DA*。这正

是 𝔟$_1$要打印的东西，它以一个冒号开头来划定每一个完全格局：

ə	ə	;		D		A		D		D		C		R		D		A		A	

;		D		A		A		D		D		C		C		R		D		A		::	

:		D		A		•••

𝔘 的下一个m-格局是 anf，唐纳德·戴维斯建议用此代表 anfang，anfang 是德语表示“开头”的单词。第一行中的函数 𝔤 在函数表中被误写作 𝔮 。它用来搜索第二个参数的最后一次出现。

anf	g(anf$_1$, :)	anf. The machine marks the configuration in the last complete configuration with y. → fom.
anf$_1$	con(fom, y)	

anf	g(anf$_1$, :)	anf，机器用y标记最后一个完全格局中的格局，然后→fom。
anf$_1$	con(fom, y)	

𝔤找到冒号（当前的完全格局之前）之后，con用字母*y*标记这个m-格局。前面为con$_1$添加的一行在这里发挥了作用：它打印一个*D*（表示一个空格），同时标记这个格，如下：

ə	ə	;		D		A		D		D		C		R		D		A		A	

;		D		A		A		D		D		C		C		R		D		A		::	

:		D	y	A	y	D	y	…

在完全格局中标记m-格局的过程中，每当con本该找到一个代表扫描符的*D*却遇到一个空格时，con$_1$就打印一个*D*。在需要的方格越来越多时，纸带就是这样逐渐增长的。

现在机器必须找到这样的指令，其格局和刚才的完全格局中标记为*y*的符号相匹配。当然，有很多指令，但是它们很容易被定位，因为每一条指令前都有分号。从最后一条指令开始向前测试这些指令。m-格局fom看上去与fom很像，但其实是kom，可能是几个表示“比较”的单词缩写形式中的一个。

fom	;	*R*, *Pz*, *L*	con(fmp, *x*)	fom. The machine finds the last semi-colon not marked with *z*. It marks this semi-colon with *z* and the configuration following it with *x*.
	z	*L*, *L*	fom	
	not *z* nor ;	*L*	fom	

fom	;	R, Pz, L	con(fmp, x)	fom，机器寻找最后一个没有被z标记的分号。它把这个分号标记上z，并且把该分号之后的格局标记为x。
	z	L, L	fom	
	not z nor ;	L	fom	

在第一趟里，fom 找到最后一个（最右边）指令后，在分号后打印一个z，然后用 con 标记紧跟着的格局。

ə	ə	;		D		A		D		D		C		R		D		A		A	

;	z	D	x	A	x	A	x	D	x	D		C		C		R		D		A		::	

:		D	y	A	y	D	y	•••

标记z表明这条指令已经检查过。在后面寻找匹配的尝试中，fom 会跳过之前已经标记为z的那些分号。

m-格局 fmp （另外一个表示“比较”的缩写？）使用 cpe 比较标记为*x*（表示一条指令中的m-格局和扫描符）和标记为*y*的格局（表示完全格局中的当前m-格局和扫描符）。

fmp	cpe(e(fom, x, y), sim, x, y)	fmp. The machine compares the sequences marked x and y. It erases all letters x and y. → sim if they are alike. Otherwise → fom.

tmp	cpe(e(tom, x, y), sim, x, y)	tmp，机器比较标记为x和y的序列。它擦除所有的x和y。如果这些序列相似就→ sim，否则→ tom。

函数cpe在比较那些标记所标识的字母时顺便擦除了这些标记。如果找到一次匹配，那么所有的*x*和*y*都被擦除，然后转向sim（表示相似）。

如果标记为*x*和*y*的格局不匹配（就像我们例子中的这样），则转向cpe的第一个参数，它是个函数 e （擦除），用来擦除其余的*x*和*y*标记，最后退回到tom来处理下一条指令。

函数tmp中有个小问题，图灵从来没有定义过包含一个m-格局参数和两个符号参数的函数 e 。而且，由于部分甚至全部的*y*标记都已经被cpe擦除，他不能回退到tom 。事实上，他需要返回到用anf再次标记这个格局。唐纳德·戴维斯提议，指令应该读取：

tmp	cpe(e(e(anf, x), y), sim, x, y)

在我们的例子中，anf_1将重新标记完全格局中的m-格局和扫描符，tom将标记下一条指令（按指令从后往前的顺序）。

ə	ə	;	z	D	x	A	x	D	x	D		C		R		D		A		A	

;	z	D		A		A		D		D		C		C		R		D		A		::	

:		D	y	A	y	D	y	…

这一次，由kmp调用的函数cpe将检测匹配，然后转向sim。所有的标记*x*和*y*都没有了，只剩下了标记*z*。最左边的标记*z*位于𝔄必须执行的指令之前。图灵如下总结了至此的进展。

> anf. Taking the long view, the last instruction relevant to the last configuration is found. It can be recognised afterwards as the instruction following the last semi-colon marked *z*. → sim

> anf，从长远角度考虑，与最后一个格局相关的最后一条指令找到了。因为这条指令跟在用*z*标记的最后一个分号后，所以今后能被识别出来。然后→sim。

事实上，这是第一个（最左边）被标记为*z*的分号，但是，是最后一条被检测的指令。m-格局sim开始使用f′在指令前的分号处寻找该标记及其位置。正如前面提到的，指令包含五个部分：m-格局，扫描符，要打印的符号，*L*、*N*或*R*，以及最终m-格局。

[245]

$\mathfrak{sim}$			$\mathfrak{f}'(\mathfrak{sim}_1, \mathfrak{sim}_1, z)$
$\mathfrak{sim}_1$			$\mathfrak{con}(\mathfrak{sim}_2,)$
$\mathfrak{sim}_2$	A		$\mathfrak{sim}_3$
	not A	R, Pu, R, R, R	$\mathfrak{sim}_2$
$\mathfrak{sim}_3$	not A	L, Py	$\mathfrak{e}(\mathfrak{mf}, z)$
	A	L, Py, R, R, R	$\mathfrak{sim}_3$

$\mathfrak{sim}$. The machine marks out the instructions. That part of the instructions which refers to operations to be carried out is marked with u, and the final m-configuration with y. The letters z are erased.

$\mathfrak{sim}$			$\mathfrak{f}'(\mathfrak{sim}_1, \mathfrak{sim}_1, z)$
$\mathfrak{sim}_1$			$\mathfrak{con}(\mathfrak{sim}_2,)$
$\mathfrak{sim}_2$	A		$\mathfrak{sim}_3$
	not A	R, Pu, R, R, R	$\mathfrak{sim}_2$
$\mathfrak{sim}_3$	not A	L, Py	$\mathfrak{e}(\mathfrak{mf}, z)$
	A	L, Py, R, R, R	$\mathfrak{sim}_3$

$\mathfrak{sim}$，机器标记出这些指令。那些表示机器必须执行的操作的指令标记为u，最终m-格局标记为y。字母z被擦除。

m-格局$\mathfrak{sim}_1$指向的是缺少第二个参数的函数$\mathfrak{con}$。实质上，它跳过m-格局和扫描符，把读写头定位在打印操作的第二个字符。

ə ə ; z D   A   D   D   **C**   R   D   A   A

; D A A D D C C R D A ::

: D A D …

m-格局$\mathfrak{sim}_2$的第二行有误：埃米尔·波斯特指出，它应该在打印*u*之前把读写头向左移。m-格局$\mathfrak{sim}_2$和$\mathfrak{sim}_3$标记这个操作（要打印的符号和读写头移动的字母）和下一个m-格局。函数$\mathfrak{e}$在转向$\mathfrak{mf}$之前擦除标识记*z*。

ə ə ;   D   A   D   D u C u R u D y A y A y

; D A A D D C C R D A ::

: D A D …

m-格局m$\mathfrak{mf}$（看起来像*mf*，实际上是*mk*，可能代表“标记”）现在标记最后一个完全格局。函数$\mathfrak{g}$（函数表里误记作$\mathfrak{q}$）的第一个参数应该是$\mathfrak{mf}_1$，而不是$\mathfrak{mf}$。

m-config.	Symbol	Operations	Final m-config.
$\mathfrak{mk}$			$\mathfrak{g}(\mathfrak{mk}, :)$
$\mathfrak{mk}_1$	not A	R, R	$\mathfrak{mk}_1$
	A	L, L, L, L	$\mathfrak{mk}_2$
$\mathfrak{mk}_2$	C	R, Px, L, L, L	$\mathfrak{mk}_2$
	:		$\mathfrak{mk}_4$
	D	R, Px, L, L, L	$\mathfrak{mk}_3$
$\mathfrak{mk}_3$	not :	R, Pv, L, L, L	$\mathfrak{mk}_3$
	:		$\mathfrak{mk}_4$
$\mathfrak{mk}_4$			$\mathfrak{con}(\mathfrak{l}(\mathfrak{l}(\mathfrak{mk}_5)),)$
$\mathfrak{mk}_5$	Any	R, Pw, R	$\mathfrak{mk}_5$
	None	$P:$	$\mathfrak{sh}$

$\mathfrak{mk}$. The last complete configuration is marked out into four sections. The configuration is left unmarked. The symbol directly preceding it is marked with x. The remainder of the complete configuration is divided into two parts, of which the first is marked with v and the last with w. A colon is printed after the whole. $\rightarrow \mathfrak{sh}$.

$\mathfrak{mf}$			$\mathfrak{g}(\mathfrak{mf}, :)$
$\mathfrak{mf}_1$	not A	R, R	$\mathfrak{mf}_1$
	A	L, L, L, L	$\mathfrak{mf}_2$
$\mathfrak{mf}_2$	C	R, Px, L, L, L	$\mathfrak{mf}_2$
	:		$\mathfrak{mf}_4$
	D	R, Px, L, L, L	$\mathfrak{mf}_3$
$\mathfrak{mf}_3$	not :	R, Pv, L, L, L	$\mathfrak{mf}_3$
	:		$\mathfrak{mf}_4$
$\mathfrak{mf}_4$			$\mathfrak{con}(\mathfrak{l}(\mathfrak{l}(\mathfrak{mf}_5)),)$
$\mathfrak{mf}_5$	Any	R, Pw, R	$\mathfrak{mf}_5$
	None	$P:$	$\mathfrak{sh}$

$\mathfrak{mf}$，最后一个完全格局被标记成四个部分。这个格局未被标记。它之前的符号标记成x。其余的完全格局分成两个部分，第一部分标记为v，第二部分标记为w。最后打印一个冒号，然后→ $\mathfrak{sh}$。

m-格局$\mathfrak{mf}$使用$\mathfrak{g}$寻找最右边的冒号。这个冒号出现在最后一个完全格局之前。完全格局出现在F-格处，通常情况下，由*D*后接0个或若干个*C*构成，每一个都表示纸带上的一个符号。这些符号里的某个地方埋藏了一个m-格局，它是表示成一个*D*后接若干个*A*的序列。

m-格局$\mathfrak{mf}_1$用来寻找埋藏在完全格局中的m-格局。当找到一个*A*时，它把读写头向左移至这个m-格局之前的格的最后一个符号处。那个格标记为*x*。接着，$\mathfrak{mf}_3$把所有前面的字符都标记为*v*。

$\mathfrak{mf}_3$遇到一个冒号时，$\mathfrak{mf}_4$就接替它开始工作。$\mathfrak{mf}_4$使用$\mathfrak{con}$跳过m-格局和扫描字符。当它找到非*C*的字符时就停下来。除了扫描字符，其他符号均标记为*w*。最后，$\mathfrak{mf}_5$打印一个冒号。

下面是一个比所分析的简单例子稍微复杂一点的完全格局：

:		D		C		D		D		A		D		C		D		D		C				…

这个完全格局表示一个以0（*DC*）和一个空白（*D*）开头的纸带。接下来的格是一个扫描格，是用格局q_1（*DA*）表示的。扫描格是一个0（*DC*），后跟一个空白（*D*）和一个0（*DC*）。$\mathfrak{mf}$结束时，纸带如下：

:		D	v	C	v	D	x	D		A		D		C		D	w	D	w	C	w	:		…

唯一未标记的是格局（它由m-格局*DA*和扫描符*DC*组成）。

在前面的简单例子里，m-格局的左边和扫描格的右边都没有符号，因此，*v*、*x*和*w*标记都没有发挥任何作用：

ə	ə	;		D		A		D		D	u	C	u	R	u	D	y	A	y	A	y

;		D		A		A		D		D		C		C		R		D		A		::	

:		D		A		D		:		…

所有的部分都已标记了。指令的操作和最终m-格局分别标记为*u*和*y*，完全格局的一部分标记为*v*、*x*和*w*。

除了因为机器扫描到一个0或1，需要再打印一遍0或1的情况，如果指令打印了一个0或1，那么通用机也需要打印一个0或1。通用机只有在扫描到空格时才打印一个0或1。这就是$\mathfrak{sh}$（可能代表“展示”）的功能。

$\mathfrak{sh}$			$\mathfrak{f}(\mathfrak{sh}_1, \mathfrak{inst}, u)$	$\mathfrak{sh}$. The instructions (marked u) are examined. If it is found that they involve "Print 0" or "Print 1", then 0: or 1: is printed at the end.
$\mathfrak{sh}_1$		L, L, L	$\mathfrak{sh}_2$	
$\mathfrak{sh}_2$	D	R, R, R, R	$\mathfrak{sh}_2$	
	not D		$\mathfrak{inst}$	
$\mathfrak{sh}_3$	C	R, R	$\mathfrak{sh}_4$	
	not C		$\mathfrak{inst}$	
$\mathfrak{sh}_4$	C	R, R	$\mathfrak{sh}_5$	
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 0, :)$	
$\mathfrak{sh}_5$	C		$\mathfrak{inst}$	
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 1, :)$	

$\mathfrak{sh}$			$\mathfrak{f}(\mathfrak{sh}_1, \mathfrak{inst}, u)$
$\mathfrak{sh}_1$		L, L, L	$\mathfrak{sh}_2$
$\mathfrak{sh}_2$	D	R, R, R, R	$\mathfrak{sh}_2$
	not D		$\mathfrak{inst}$
$\mathfrak{sh}_3$	C	R, R	$\mathfrak{sh}_4$
	not C		$\mathfrak{inst}$
$\mathfrak{sh}_4$	C	R, R	$\mathfrak{sh}_5$
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 0, :)$
$\mathfrak{sh}_5$	C		$\mathfrak{inst}$
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 1, :)$

$\mathfrak{sh}$，表示检测标记为u的指令。如果指令中包含“打印0”或“打印1”，那么在末尾打印0:或1:。

首先，$\mathfrak{sh}$定位到最左边的标记u，$\mathfrak{sh}_1$将读写头向左移动三个位置，置于代表扫描格的最后一个符号处。如果扫描的格是一个空格，那么那个符号可以是一个*D*。如果不是*D*，那么将跳过其余的m-格局转向$\mathfrak{inst}$。

如果扫描的字符是一个空格，则$\mathfrak{sh}_2$转向$\mathfrak{sh}_3$（不是表中指示的$\mathfrak{sh}_2$），接着检查$\mathfrak{sh}_3$、$\mathfrak{sh}_4$和$\mathfrak{sh}_5$，看打印的指令是否为*DC*（表示打印0）或者*DCC*（表示打印1）。如果是，那么$\mathfrak{pe}_2$在纸

带的尾部打印出相应的数字和一个冒号。现在的示例纸带如下：

ə	ə	;		D		A		D		D	u	C	u	R	u	D	y	A	y	A	y

;		D		A		A		D		D		C		C		R		D		A		::	

:		D		A		D		:		0		:		...

显然，表的 $\mathfrak{sh}$ 部分因为使用了二进制数而非十进制数而得以简化。十进制数要求另外8个m-格局（$\mathfrak{sh}_6$到$\mathfrak{sh}_{13}$）来打印2～9位。

无论打印了0或1，或者什么都没打印，通用机都转向 $\mathfrak{inst}$ [可能表示instruction（指令），不过表示为instigate（鼓动），更生动]。最后剩下的工作就是实现 $\mathcal{M}$ 的下一个完全格局。它包括当前格局里标记为*x*、*v*和*w*的所有符号，因为这些符号保持不变。然而，m-格局和扫描的格会被替换。它们将被标记为*y*的m-格局和标记为*u*的符号替代。

$\mathfrak{inst}$ 的表中又一次引用了最初被定义为 $\mathfrak{q}$ 的函数 $\mathfrak{g}$ 。同样，第五行的函数 $\mathfrak{ec5}$ 同第三行和第四行中的 $\mathfrak{ec5}$ 一样。

[246]

$\mathfrak{inst}$			$\mathfrak{g}(\mathfrak{l}(\mathfrak{inst}_1), u)$	$\mathfrak{inst}$. The next complete configuration is written down, carrying out the marked instructions. The letters u, v, w, x, y are erased. → $\mathfrak{anf}$.
$\mathfrak{inst}_1$	α	R, E	$\mathfrak{inst}_1(\alpha)$	
$\mathfrak{inst}_1(L)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{inst}_1(R)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, x, u, y, w)$	
$\mathfrak{inst}_1(N)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, x, y, u, w)$	
$\mathfrak{ov}$			$\mathfrak{e}(\mathfrak{anf})$	

$\mathfrak{inst}$			$\mathfrak{g}(\mathfrak{l}(\mathfrak{inst}_1), u)$	$\mathfrak{inst}$，写入的下一个完全格局，用于执行被标记的指令。字母u、v、w、x、y被擦除。最后→$\mathfrak{anf}$。
$\mathfrak{inst}_1$	α	R, E	$\mathfrak{inst}_1(\alpha)$	
$\mathfrak{inst}_1(L)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{inst}_1(R)$			$\mathfrak{ce}_5(\mathfrak{ov}, v, x, u, y, w)$	
$\mathfrak{inst}_1(N)$			$\mathfrak{ec}_5(\mathfrak{ov}, v, x, y, u, w)$	
$\mathfrak{ov}$			$\mathfrak{e}(\mathfrak{anf})$	

事实上，函数$\mathfrak{ce}_4$和$\mathfrak{ce}_5$都没有定义。基于$\mathfrak{ce}_3$，我们可以很容易地构造它们：

$\mathfrak{ce}_4(\mathfrak{B}, \alpha, \beta, \gamma, \delta)$	$\mathfrak{ce}(\mathfrak{ce}_3(\mathfrak{B}, \beta, \gamma, \delta), \alpha)$
$\mathfrak{ce}_5(\mathfrak{B}, \alpha, \beta, \gamma, \delta, \varepsilon)$	$\mathfrak{ce}(\mathfrak{ce}_4(\mathfrak{B}, \beta, \gamma, \delta, \varepsilon), \alpha)$

函数$\mathfrak{ce}_5$把标记为α的符号顺序地复制到纸带的尾部，然后再复制标记为β的字符，如此等等，在复制的过程中擦除这些标记。

m-格局$\mathfrak{inst}$引用$\mathfrak{g}$。$\mathfrak{g}$寻找最右边的标记为u的符号，该符号可以为L、R或者N。m-格局$\mathfrak{inst}_1$扫描并擦除那个符号，然后根据此符号分别转向$\mathfrak{inst}_1(L)$、$\mathfrak{inst}_1(R)$或者$\mathfrak{inst}_1(N)$。图灵的用意很明显，不过事实上，我必须阻止在机器的这个点上引入新的语法，更何况这是没有必要的。我们把整个的$\mathfrak{inst}_1$格局替换成如下的表达式：

$\mathfrak{inst}_1$	L	R, E	$\mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w)$
	R	R, E	$\mathfrak{ce}_5(\mathfrak{ov}, v, x, u, y, w)$
	N	R, E	$\mathfrak{ce}_5(\mathfrak{ov}, v, x, y, u, w)$

在这三种情况下，标记为v的格首先被复制到纸带的尾部，最后复

制标记为*w*的格。由*v*标记的这些符号都处于完全格局左边，直到（但不包括）被扫描的格的左边。那个格标记为*x*。标记为*w*的符号都处于被扫描的格右侧。

ec5中间的这三次复制取决于读写头是左移、右移或不移动。顺序如下。

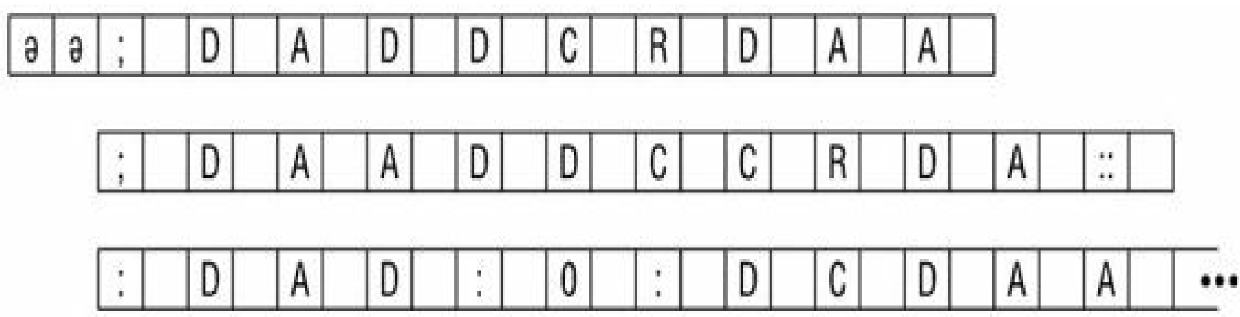
左移：下一个m-格局/读写头左边的符号/被打印的符号

右移：读写头左边的符号/被打印的符号/下一个m-格局

不移动：读写头左边的符号/下一个m-格局/被打印的符号

例如，如果读写头向左移动，那么下一个m-格局插入到上一次读写头左边格的前面。如果读写头向右移动，则下一个m-格局放置于被打印符号的右边。

ec5的每一个函数都要转向ov [可能代表over（结束）]。函数e 擦除所有E-格，然后在下一次移动之前转向anf 。纸带现在的情况如下：



第二个完全格局包含了符号*DC*（代表0），后跟*DAA*，它表示一个新的m-格局q_2。

图灵在定义通用机时是有一些限制的，它不能模仿任意的普通图灵机，因为它不能在完全格局的左边插入空白，因此模仿任意一个读写头移向初始位置左边的机器，它都不能正确的工作。（确实，图灵在函数con中省略了向右边插入空白的过程。）这样的通用机也只能在模仿用0或1代替空白，且这种代替按从左向右的统一方式进行的机器时正确运行。如果机器不是这样操作的，那么通用机也可以模仿它，只是不能打印出0和1的正确序列。

尽管存在这些限制，还有一些小的打印问题和错误，但是图灵还是做出了杰出的贡献。他通过展示这样一台可以适当地编程来执行任意计算机器操作的通用机，论证了计算的一般性。一本受到称赞的关于可计

算性的书中这样写道："图灵关于通用图灵机存在性的理论是上世纪的一个思想路标。"[4]

所有这些都引发了一个问题：

阿兰·图灵发明过计算机吗？

---

[1] 出现在埃米尔·波斯特论文"Recursive Unsolvability of a Problem of Thue"，*The Journal of Symbolic Logic*，Vol. 12，No. 1（Mar. 1947），1-11的附录中。整篇文章转载于Martin Davis，ed.，*The Undecidable*（Raven Press，1965），293-303。附录转载于B. Jack Copeland，ed.，*The Essential Turing*（Oxford University Press，2004），97-101。

[2] 唐纳德·戴维斯的"Corrections to Turing's Universal Computing Machine"，in C. Jack Copeland，ed.，*The Essential Turing*，103-124。任何想要编写仿真通用机程序的人都想专研戴维斯的文章。

[3] 由波斯特建议，"Recursive Unsolvability of a Problem of Thue"，7。

[4] John P. Burgess在George S. Boolos、John P. Burgess和Richard C. Jeffrey所著的Computability and Logic, fourth edition（Cambridge University Press，2002）一书的前言中提到的。

第10章 计算机与可计算性

想像一台计算机器，它几乎什么也做不了，图灵事实上是构想了一种用于“通用目的”的多功能计算机，这是一个革命性的观念。在当时，计算机被普遍认为是为某些特定工作特殊设计的。早期类似于计算机的设备，如微分分析仪（由麻省理工学院的万尼瓦尔·布什及其学生设计并构建），就是这一设计方式的例证。微分分析仪具有重要的功能——求解常微分方程，但它所能做的也仅限于此。

即使是那些深入探索构建数字计算机的人们，也很难抓住数字逻辑的普遍规律。如霍华德·艾肯，他从1937年就开始研究数字计算机，是计算机行业一位真正的先驱。然而在1956年，他说道：

> 如果事实证明，一台用来求微分方程数值解的机器的基础逻辑，与一台为百货店制作账单的机器的逻辑相同，那么这将是我遇到的最令人惊讶的巧合。[1]

作为一个将计算机视为逻辑机器的人，图灵显然知道的更清楚。当大部分早期的计算机制造者在考虑硬件问题的时候，图灵从1936年起就开始编写软件了。对于图灵来说，即使是加法这样的基本算术运算也可以通过软件来实现。与艾肯在1956年的言论不同，图灵在1950年这样说道：

> 数字计算机的特殊性在于它们能够模拟任何离散状态的机器。为描述这一点，我们称之为通用机。有了具备这一特性的机器，我们就能得到一个重要的结果：如果不考虑速度，那么就不需要对不同的计算过程设计不同的机器。通过对每个事件恰当编程，所有的工作可以在一台机器上完成。在这种意义下，所有的数字计算机都是等价的。[2]

图灵引入了一个重要的限制“如果不考虑速度”。有些人也许会说，对于计算机而言，速度不是全部，却不可或缺。当人们需要某种特殊的

计算机时，例如为数百万的好莱坞大片做电脑成像，速度通常是主要的考虑因素，而强大的存储能力反而不重要。然而，在实际的数字运算能力上，所有的数字计算机都是通用的。

在关于计算的发展史上，阿兰·图灵的地位从来都不十分明确。在一部权威发展史著作[3]中，图灵的贡献几乎没有提到；而在另一位著名数学家写的历史中，计算机被看成是数学概念的物理体现[4]，图灵就被视为一个最重要的人物。图灵在计算历史著作中的地位取决于到底是从工程和商业的角度，还是从数学和学术的角度来看待计算机。

图灵一个令人感兴趣的角色涉及图灵和约翰·冯·诺依曼的关系。他们的第一次见面是在1935年4月，当时冯·诺依曼从普林斯顿大学来到剑桥大学讲授关于殆周期函数的课程。在此之后，图灵决定加入普林斯顿大学。[5] 1936年秋，图灵到达普林斯顿大学后，他们又取得了联系。[6]冯·诺依曼一度声称，在哥德尔的不完备性定理之后便不再读数理逻辑方面的论文了，[7]所以我们并不知道他到底什么时候读了“可计算数”那篇论文。这两位数学家也有其他共同的数学兴趣（殆周期函数和群论），这些是冯·诺依曼在1937年6月1日为图灵写的关于普罗科特奖学金的推荐信中提及的。

1938年7月，图灵离开普林斯顿之前，冯·诺依曼为他提供了一份年薪1500美元的工作，是在高级研究院担任自己的助手，但是图灵没有接受。当时，冯·诺依曼必然已经读过了图灵的论文。在图灵传记中，安德鲁·霍奇斯向物理学家斯坦尼斯罗·乌拉姆（当时也在高级研究院）询问冯·诺依曼对图灵的评价（冯·诺依曼逝于1957年，时年53岁）。乌拉姆回忆起1938年夏天同冯·诺依曼一起旅行时，当时冯·诺依曼建议玩一个游戏。

> 在一张纸上写下一个尽可能大的数，用一种与图灵模式有关的方法来定义它…冯·诺依曼对他十分赏识，早在1939年就和我提到他的名字，并且说这是一个“聪明的主意”……1939年，冯·诺依曼在考虑用机械方法来研究形式化数学系统时，曾数次向我提到图灵的名字。[8]

早年和图灵之间的这些接触，在1944年9月冯·诺依曼到达宾夕法尼亚大学的穆尔电气工程学院时突然变得非常重要。当时正在建造的计算机叫作ENIAC（电子数字积分计算机）。这个由约翰·埃克特（1919—1995年）和约翰·威廉·莫奇利（1907—1980）设计的30吨的庞然大物，虽然建造出来了，但是它的局限性越来越明显。于是一个叫EDVAC（电子离散变量自动计算机）的替代品列入了后续计划之中。

从一开始，冯·诺依曼就不单单从潜在用户，也从科学家和技术人员的角度来考虑问题。在1944年剩下的时间以及整个1945年，他并不在洛斯阿拉莫斯国家实验室，他参加了EDVAC方面的技术会议，做出了很多技术贡献，并且在逻辑设计方面提出了很多建议。[9]

然而，1945年6月30日，当一篇以约翰·冯·诺依曼为唯一作者的“EDVAC的第一份草案报告”[10]出现时，争议的火焰被点燃了，即使在今天仍然余烟未尽。这篇报告强调了一些概念，比如计算机应该是电子的，必须使用二进制数，以及程序应该存储在内存中。但是也许再也无法完全确定，这些概念是冯·诺依曼自己提出的，还是他只不过是将在穆尔学院构建ENIAC时的想法组合了一下。数十年来，当人们描述数字计算机的时候总是会提到“冯·诺依曼体系结构”，现在这个词渐渐地不再使用，很大程度是因为这样称呼缺乏对在计算机体系结构上有所贡献的其他人的尊重。

“EDVAC的第一份草案报告”只引用了一篇出版文献，它是发表在期刊《数学生物物理学公报》上题为“神经活动内在概念的逻辑演算”[11]的论文。这篇引用文献表明了冯·诺依曼在计算机和人脑关系上的兴趣，同样有趣的是，这篇论文的作者所提出的大脑的生理学概念正是基于图灵机的方法。麦卡洛克和匹茨的论文同样被诺伯特·维纳（1894—1964）的经典著作《控制论：关于在动物和机器中控制和通信的科学》（1948）所引用。我会在第17章详细讨论麦卡洛克、匹茨、维纳和冯·诺依曼。

物理学家斯坦利·弗兰克尔曾在洛斯阿拉莫斯与冯·诺依曼一起工作，他还记得冯·诺依曼对于图灵1943年（或1944年）发表的那篇论文是如此地充满兴趣：

> 冯·诺依曼向我介绍了那篇论文，在他的催促下我认真地阅读了论文。很多人都称冯·诺依曼是“计算机之父”（以现在对这个词的理解），但是我保证他自己从未犯过这样的错误。他一直和我（肯定还有别人）强调，这些基础概念都是属于图灵的（图灵达到了巴贝奇、埃达和其他人都预想不到的高度），而他只是起了助推的作用。在我看来，冯·诺依曼的本质角色就是，让世界知道图灵的这些基础概念，并在穆尔学院和其他地方进行了研究工作。[12]

20世纪40年代末，冯·诺依曼似乎已经向一些人提到图灵工作的重要性。例如，1946年，他在给诺伯特·维纳的信中写道：“图灵的一个伟

大贡献……，明确了一种机制可以是‘普遍的’。”[13]

虽然阿兰·图灵多数情况下是因为他的论文而被人们铭记的，但是他的名字同样与三个大型计算机项目相联系。

第一个项目是巨人（Colossus），1943年制造于布莱切利庄园的破译密码的计算机。它是由托马斯·弗劳尔斯（1905—1998）设计的，20世纪30年代他在英国邮政电话分部的研究实验室工作，熟悉开关电路和电子学。虽然一些人认为图灵也参加了巨人项目[14]，但是很显然，他并没有。他当然知道这个项目，但是“拒绝受邀负责其中一个主要部分”[15]。尽管如此，图灵论文对巨人的逻辑设计的影响是公认的。[16]

图灵更多参与的是位于伦敦西南部特丁顿的国家物理实验室（NPL）的计算机项目。1944年，NPL的领导者是查尔斯·达尔文爵士（1887—1962）[17]，他的祖父曾撰写了在生物学上颇具影响的著作。达尔文创建了一个“数学部”，其工作就是研制自动计算机器。

数学部的负责人J. R. 沃默斯利在1945年6月招图灵来NPL面试。[18]沃默斯利读过那篇“可计算数”，并且希望图灵设计一台“自动计算引擎”（ACE）的计算机，而“引擎”这个词有意无意中唤起了他对查尔斯·巴贝奇的回忆。

图灵那时已经读过了冯·诺依曼那篇关于EDVAC的报告，对他自己的计算机有了一些想法，并在1945年结束之前完成了报告“在数学部中开发自动计算引擎的方案”。图灵的报告虽然说“对提出的计算器有了十分全面的考虑”，但还是建议“与冯·诺依曼‘关于EDVAC的报告’一起阅读”。[19]

图灵提出的机器是电子的，使用二进制数字（虽然位和位需要连续传输），具有1MHz的时钟速率。它使用水银延迟线存储器，通过水银管中的声波脉冲来存储比特信息。一个五英尺长的水银管可以存储1024比特的数据。每一比特从水银管的一端传送到另一端大约需要1毫秒，这样它可以被存取并返回水银管的开始再利用。图灵预计，一个32比特的数的加法需要32微秒（每一比特一微秒），而32比特的数的乘法需要“超过2毫秒”的时间。[20]

图灵的设计只用了很少的原始指令，大部分可以在寄存器和内存之间传输。这种情况下，它类似现代的精简指令集计算机（RISC）——采用较快的硬件，更复杂的工作由软件完成。图灵似乎还发明了栈——最终成为了计算机存储的常用类型，类似于自助餐厅里用弹簧托架摞得很高的一堆盘子。最后一个盘子“推”进栈里，成为下一个要被“弹”出的盘子。图灵将这两个方法分别叫“盖住”（BURY）和“掀开”（UNBURY）。[21]

图灵在1947年2月20日给伦敦数学协会写的讲稿中，对计算给出了

更具个人观点的说法："类似ACE这样的计算机器……实际上是更加实际的通用机器。"这个机器必须做的工作的复杂性就是"要专注于纸带"（其实就是软件），"而且不应该以任何方式出现在通用机器中"。[22] 图灵认识到了计算机速度的重要性，当然，他还强调了大存储的好处：

> 我相信提供适当的存储器是解决数字计算机问题的关键。如果想让计算机可以拥有人工智能这一说法更具有说服力，必须提供比现在多得多的存储能力。在我看来，制造大的内存要比用更快的速度计算诸如乘法的运算重要得多。[23]

正如所料，图灵清楚地认识到了二进制相比于十进制而言所拥有的优势：

> 对于大规模计算机而言，使用二进制是很自然的事情。在二进制范围内，计算更简单，因为制定只有两个固定位置的机制很容易。[24]

在讲稿的最后，图灵特别提到了可以修改自己指令集的机器：

> 就像一个小学生，他从老师那里学到了很多东西，同时通过自己的学习也增长了很多知识。我觉得，当理应把这台机器看作是在展示智能的时候，就会是这种情况。[25]

到了1947年9月，ACE缺乏进展开始令图灵感到沮丧。他请了一年的半薪休假，离开了剑桥。NPL期望图灵至少能再回来工作两年，但是他没有回来。（实验版的ACE一直到1950年才就绪，而且已经和图灵当初的设计偏离了许多。）

而图灵加入了从1945年就在曼彻斯特大学的麦克斯·纽曼的队伍里。纽曼获得批准，建立了一个新的计算机器实验室，并制造了一台叫做Mark I的机器。在1948年6月，Mark I成为了"第一台完工的EDVAC类型的电子程序存储计算机"。[26]

图灵在9月加入了曼彻斯特大学的数学系，参与纽曼的新项目。两个月后，他们和曼彻斯特的一个电子制造商费伦蒂有限公司达成协议，为后者制造商业化的机器。

图灵基本上是负责Mark I的编程工作。大约1951年，图灵接受的任务是为这个机器产品编写第一本"程序员手册"。在手册中，图灵将编程定义为："一种使数字计算机按照人的意愿工作，并将其正确表达在穿

孔纸带上的活动。”[27]

除了使用水银延迟线，Mark I还使用了阴极射线管（CRT）来做存储器。这种存储器一般称为威廉姆斯管，是1946年12月来到曼彻斯特的F. C. 威廉姆斯率先研究出来的。需要存储的数据以电脉冲形式发送到CRT，然后用一组点阵显示在屏幕上，每个点代表一个比特。用不同密度或大小的点来区分0和1。电子管前的金属盘用来接收这些点，并且允许电子管进行刷新或者读取。另一个CRT允许人们来查看这些点并且检查数据。到了1947年，每个CRT都可以存储2048比特的数据。

Mark I用40比特的字来存储数据，这样可以存储两条20比特的指令。这些字以5比特一组显示在CRT上，其中每个5比特码由对应这个码的电传打印机字符表示，由此开发了一种32进制的记法。为了读一个数字，需要知道每个字符对应的码。这些字符码并不是以字母顺序排列的，按照图灵的做法，每个想要在Mark I上开发程序的人都需要记住这个32字符的序列：

$$/E@A:SIU\ \frac{1}{2}\ DRJNFCKTZLWHYPQOBG"MXV£$$

图灵参加这些实际的计算机项目可能会让我们忽略一个简单的事实：图灵“可计算数”论文的目的并不是设计一个通用的计算机器。这篇论文的全部目的是希望用这个假定的计算机来帮助解决判定性问题。这里还有几个步骤要做。一个关键的步骤就是要证明，图灵机本质上会被限定在它能做的事情中。

图灵在论文的引言中提到：“尽管可计算数如此之多，并且在很多方面与实数相似，但它是可数的。”（论文第230页，本书第57页。）在第5节中，他证明了“每个可计算序列至少对应一个描述数，但不存在一个描述数对应多个可计算序列。因此，可计算序列和可计算数是可数的。”（论文第241页，本书第125页。）

仍有一些疑问。很显然，可计算数很多，至少包含一些超越数。用图灵机计算π、e或者刘维尔常数是完全可能的。确实，数位之间符合某种次序的超越数是图灵机可计算的，这就是乌拉姆和冯·诺依曼在一起旅行时玩的游戏。

然而，大部分……不不不，我们现实点，就说“事实上全部”吧。事实上，全部的超越数都貌似其数位是随机的。在实数领域，数位序列的顺序性或者可计算性并不怎么提及，常提到的是完全性和总体随机性。

你如何制造一台可以计算一个没有规律的数的机器呢？你只是随机地生成数位吗？

随机性不是计算机能处理好的事情，但是计算机经常要求一些随机

的行为。一些统计程序需要随机数，一些电脑游戏需要随机数来改变行为。没有随机数，纸牌游戏的每手牌都会完全一样。

程序语言总是为程序员提供一些标准方法来生成随机数。例如，用C语言写的计算机程序可以使用叫做rand的函数来获得一个0到32767之间的随机数。rand函数从一个叫做种子的数开始，种子初始化默认为1。不同的rand函数可能以不同的方式来实现算法。例如，下面是rand函数的微软C语言[28]实现：

```
int seed = 1;

int rand ()
{
  return ((seed = seed * 214013 + 2531011) >> 16) & 32767;
}
```

这个rand函数将种子乘以214 013，然后加上2 531 011，再后用这个结果替换种子，供rand函数下次调用。然而，种子被定义为32位有符号整数，可能会出现向上溢出或者向下溢出。计算的结果会截位至32位，如果最高位是1，那么结果实际上是负的。[29]计算过程接着将结果移动16位，相当于将它除以65 536，然后截掉多出来的。最后在结果和32 767的位之间进行AND布尔运算。这样会去掉除低15位外的所有位，确保结果在0到32 767之间的。

即使你不理解上述费解的计算，也能看出rand函数其实不会真的生成随机数！这个函数是完全确定的。从种子值为1开始，重复地调用函数总是产生一系列相同的计算结果：

41
18 467
6 334
26 500
19 169
…

程序第一次调用rand，函数会返回41，而第30 546次调用rand，函数还是会返回41，然后一直重复。

这样的结果序列完全取决于种子和算法，它并不是真正随机的，它被称为伪随机序列。如果在rand函数生成的数字上进行统计实验，它们会表现出随机性。在一些应用程序中，伪随机序列可能比真正的随机数更好，因为它可以重新产生结果来测试程序是否正常运行。

然而，在游戏中并不希望看到总是产生相同的一串随机数。鉴于

此，每次处理纸牌游戏中的一手新牌时，程序可能会获得当前的时间，并精确到秒和毫秒，然后用此来设定新的种子。假设你不会在一天里的同一个时间（精确到毫秒）同时处理两手牌，那么你得到的牌看起来就是随机的。

约翰·冯·诺依曼曾经说："任何想用算术方法来产生随机数的人都是有罪过的。"[30]（这在他描述了用算术方法来产生随机数之前是正确的。）因为计算机是没有能力产生随机数的，如果应用程序真的需要一个随机数，那么就必须抛开计算机用特定的硬件来做这件事。硬件随机数产生器（RNG）可以使用环境噪声或者量子过程来产生随机数。

令人好奇的是，阿兰·图灵似乎想到了在硬件上生成随机数的方法。图灵提出，在曼彻斯特大学研制的Mark I的生产模型中要有一条特别的指令：从噪声源生产随机数。结果，随机数并没有预想的那么随机，不过已经足够让你不必像通常那样去调试它了。[31]

我们假设有一个硬件RNG，用它来产生一个0和1的序列，就像图灵机那样。在序列前放一个二进制小数点，你会看到一个实数（毫无疑问是超越数）在你面前被正确构造了出来。（如果你使用软件来生成伪随机序列，种子最终会被重新计算，序列会重新开始。得到的数字会由一些重复的数位序列组成，这意味着它是一个有理数。）

现在定义一个图灵机，它生成的实数与硬件RNG生成的实数完全相同。但是你做不到。复制RNG的唯一方法是图灵机显式地打印每个你想要的数位，但这不是一个拥有有限个格局的图灵机。

也许大多数实数的随机性只是一个幻想。毕竟，π的数位看起来是随机的，但π肯定是可计算的。也许表现出随机性的实数确实有一些我们不知道的底层结构。我们换一个思路考虑这个问题，也许可以证明可计算数是不可数的。然后，我们可以睡个好觉，因为知道了每个实数都是可计算的，我们很欣慰。

图灵需要直面这些挑战。

8. *Application of the diagonal process.*

It may be thought that arguments which prove that the real numbers are not enumerable would also prove that the computable numbers and sequences cannot be enumerable*. It might, for instance, be thought that the limit of a sequence of computable numbers must be computable.

* *Cf.* Hobson, *Theory of functions of a real variable* (2nd ed., 1921), 87, 88.

8. 对角线法的应用

人们可能认为，证明实数是不可数的论据同样可以用于证明可计算数及可计算序列是不可数的*。例如，可以认为一个可计算数序列的极限一定是可计算的。

* *Cf.* Hobson， *Theory of functions of a real variable*（2nd ed.，1921），87，88。

图灵暗指了格奥尔格·康托尔的第一个关于实数是不可数的证明（1874年），我在本书第20页描述过这个证明。似乎图灵当时并未读过康托尔最初的著作，于是他参考了一本由剑桥大学出版社出版[32]的E. W. 霍布森的书。霍布森的表述与康托尔的十分相似，甚至使用的记号都大部分相同。

图灵认为，使用一系列可计算数而非实数，康托尔的工作是可以重复的。在这两种情况下，数都趋于一个极限。在康托尔的证明中，该极限必然是一个实数（还有其他可能吗？），但是康托尔证明了该极限不在可数的实数集中，并由此证明了实数是不可数的。

当同样的方法用于可计算数时，可计算数列同样趋于一个极限，这个极限仍然是可计算数吗？图灵回答道：

This is clearly only true if the sequence of computable numbers is defined by some rule.

显然，只对在某种规则下定义的可计算数序列成立。

图灵所指的序列是趋于极限的阿尔法或贝塔序列。仅当我们可以对其进行计算时，该序列的极限是一个可计算数，即我们设计一个算法来得到这些阿尔法和贝塔序列接近的数值极限。这看起来不太可能。如果我们没有办法计算这个极限，那么它便不是一个可计算数，只是另一个不可计算的实数，于是我们便无法证明可计算数是可数的。

Or we might apply the diagonal process.

或者我们可以尝试对角线法。

图灵把余下的段落用引号括了起来，看起来像是一位反对者试图在他的论文里说服读者可计算数是不可数的。比起我在第23页提及的，图灵的这个反对者试图找寻一种充斥着大量记号的对角线法来解决问题。

“If the computable sequences are enumerable let α_n be the n-th computable sequence, and let $\phi_n(m)$ be the m-th figure in α_n.

“假设可计算序列是可数的，令α_n为第n个可计算序列，$\phi_n(m)$为α_n中的第m个数。

这些只是记号。每一个可计算序列都是0和1的组合，而且这样的二进制位的每一个都可以用希腊字母ø表示。可计算序列可用下标表示成

$$
\begin{aligned}
\alpha_1 &= \phi_1(1)\ \ \phi_1(2)\ \ \phi_1(3)\ \ \phi_1(4)\ \ \ldots \\
\alpha_2 &= \phi_2(1)\ \ \phi_2(2)\ \ \phi_2(3)\ \ \phi_2(4)\ \ \ldots \\
\alpha_3 &= \phi_3(1)\ \ \phi_3(2)\ \ \phi_3(3)\ \ \phi_3(4)\ \ \ldots \\
&\ldots
\end{aligned}
$$

Let β be the sequence with $1 - \phi_n(n)$ as its n-th figure.

令β是以$1-\phi_n(n)$为其第n个数的序列。

换句话说，β是翻转0和1后得到的对角线。

$$\beta = (1 - \phi_1(1))\ (1 - \phi_2(2))\ (1 - \phi_3(3))\ (1 - \phi_4(4))\ldots$$

Since β is computable, there exists a number K such that $1 - \phi_n(n) = \phi_K(n)$ all n.

由于β是可数的，则存在数K，使得$1 - ø_n(n) = ø_K(n)$对任意n成立。

即对于某个K，在所列举的可计算数中存在α_K，使得

$$\beta = \alpha_K = \phi_K(1) \quad \phi_K(2) \quad \phi_K(3) \quad \phi_K(4) \ \ldots$$

一般地， 对于n, 有$1-\phi_n(n) = \phi_K(n)$ 或 $1 = \phi_K(n) + \phi_n(n)$。

图灵的反对者使用这一算术论据来证明β不可能存在：

Putting n=K,

令，n=K,

即$1 = \phi_K(K) + \phi_K(K)$。

we have $1 = 2ø_K(K)$, *i.e.* 1 is even. This is impossible. The computable sequences are therefore not enumerable."

我们有$1=2ø_K(K)$，即1是偶数。而这是不可能的，因此可计算序列是不可数的。”

这相当有趣，这位神秘的反对者描述了如何基于所枚举的可计算序列计算一个称为β的数，而这个可计算数并不在所枚举的序列中。因此，反对者声称，可计算序列是不可数的。

但是图灵很冷静，他说：

The fallacy in this argument lies in the assumption that β is computable.

这个证明的谬误在于假设β是可计算的。

谬误？什么谬误，β怎能是不可计算的？β是由可计算序列的枚举计算得到的，那么它必然是可计算的，不是吗？

好吧，其实不是的。

让我们退一步看，图灵最初将可计算数定义为那些可以用有限方法计算的数。他建造了虚构的机器来计算这些数，并说明每台机器都可被一个称为描述数的正整数唯一标识。因为整数是可数的，图灵机是可数的，所以可计算序列是可数的。

在某种意义上，可以像枚举正整数一样简单地枚举图灵机：

1

2

3

4

5

…

所有的图灵机都将出现在描述数的列表中，并且可以从描述数中得到标准描述，这样我们可将其提供给通用机来取得可计算序列。

当然，我们还是遗漏了一些东西：我们没有方法来精确地确定列表里的哪些正整数是非循环机的描述数。

想想第2节中的定义，非循环机是指永远不停地打印0和1的机器。虽然一个永不停歇的机器看起来像是“失去了控制”或“疯狂的”，但是非循环机对于计算无理数或者有理循环数是非常必要的，甚至对于像1这样的有理数（在二进制看来是与1/2等价的），人们更加偏向于非循环机打印1及一串连续的0，

.10000000…

另一方面，循环机指的是陷入了不期望循环的机器。例如，循环机可能在不移动读写头的情况下持续打印0，也可能持续地打印除0和1之外的符号。

术语“非循环”和“循环”并非最佳的描述用语，非循环机可能无止境地处于打印0或1的小循环中，这是没问题的。而循环机可能由于转向一个不存在的m-格局而阻塞，这只是循环机会发生的诸多问题中的一种。

我们需要确定非循环机的描述数，因为它们是唯一能被通用机解释的。我们可能已经成功枚举了所有的图灵机（在正整数的区间中），但是还未确认它们是非循环的，因此不能用它们来生成可计算序列。

显然，许多整数不是机器的描述数。无论如何，我们可以很容易地确定（通过人工观测或图灵机）某个整数是不是一个结构良好的描述

数，即它可以分割成结构良好的指令集，任一指令都以m-格局作为开始。我们甚至可以确定一个机器是否引用了不存在的m-格局，同时容易地检查某个m-格局是否包含了打印0或1的指令。通过这样的过程，我们可以知道最小的结构良好的描述数为31 334 317，并且这是一个循环机（只打印空格）。第一个非循环机在313 324 317处出现，而到了313 325 317才出现了第一个从左往右打印的非循环机。

前两个非循环的、从左往右打印的图灵机在正整数枚举最开始时就确认了。

1

2

3

4

5

…

313 325 317 ←从左往右打印0的机器

…

3 133 225 317←从左往右打印1的机器

…

当然，这些是最最简单的机器，标识它们的方法也非常简单。更有难度的（实际上像图灵将要说明的，几乎不可能的）是，通过实现了某个通用过程的机器，来判断特定的整数是否是非循环机的描述数。

这个通用过程正是对角线方法的过程。β的每一位都基于不同的可计算数，因此计算β需要标识所有的非循环图灵机。图灵将证明，这些非循环机不能被有限方法标识，即不能明确地枚举可计算数序列，因此β便不是一个可计算序列。

> It would be true if we could enumerate the computable sequnces by finite means, but the problem of enumerating computable sequences is equivalent to the problem of finding out whether a given numer is the D.N of a circle-free machine, and we have no general process for doing this in a finite number of steps.

> 如果我们能够用有限步骤枚举可计算序列，那它便是成立的。然而，枚举可计算序列的问题和找出给定的数是否是非循环机的标准描述的问题是等价的，而我们没有任何通用过程能够在有限步内

处理这一问题。

图灵微妙地转移了焦点。他试图从将康托尔的对角证明法应用到可计算序列开始，然而现在，他只是想要讨论在试图标识所有非循环机的描述数时会发生什么。

> In fact, by applying the diagnoal process argument correctly, we can show that there cannot be any such general process.

> 实际上，通过正确地使用对角线法，我们能够证明不存在任何这样的通用过程。

如果像图灵断言的那样，没有任何通用过程用于确定某个整数是否是非循环机的描述数，那么β是不可计算的。这就使得反对者所谓的可计算序列不可数的证明无效了。没有什么可以削弱我们对于可计算序列实际上是可数的信心，这也就表明了可计算数无法包含所有实数。

遗憾的是，图灵在下一段的开始说得非常含糊：

> The simplest and most direct proof of this is by showing that, if this general process exists, then there is a machine which computes β.

> 关于这个命题最简单、最直接的证明是，说明如果存在这样的通用过程，那么就存在一台可以计算β的机器。

我认为他的意思是，不可能存在通用过程来确定某台机器是非循环机，因为如果存在，我们就能够计算β。而现在我们知道不可能计算β，否则对角证明法就有效了，于是可计算序列将是不可数的。

> This proof, although perfectly sound, has the disadvantage that it may leave the reader with a feeling that “there must be something wrong”.

这个证明虽然看起来很完美，但有个缺点，它很可能使读者产生“一定有什么地方出了问题”的想法。

这个悖论始终与我们的直觉格格不入，因此，图灵将要用更直接的方法来证明，并不存在一个机器能够确定某个正整数是非循环机的描述数。

The proof which I shall give has not this disadvantage, and gives a certain insight into the significance of the idea “circle-free”.

我将要给出的证明没有这方面的问题，同时会给出“非循环”这一概念更为明确的见解。

他可能在这个数论的小小练习中引入了更为深远的东西。

现在图灵想要的是一台能从可计算序列中得到一个数位的机器，他不需要考虑减去这一数位。实际上，图灵试图计算一些比β简单得多的东西。

It depends not on constructing β, but on constructing β', whose n-th figure is $\phi_n(n)$.

它并不依赖于β的构造，而是依赖β'的构造，这里β'的第n个量是$\phi_n(n)$。

在图灵论文的开始（本书第57页），他说：“尽管如此，可计算数并不完全包括所有可定义的数，我们将给出一个可定义的不可计算的数。”β和β'都是可定义的数。β'可定义，是由于我们可以给出如何计算它的指令：从1开始枚举所有的数。对每一个数，判断它是否是一个结构良好的图灵机的描述数；如果是，判断这个机器是否是非循环的；如果是，计算这个数直到它的第n位（这里n是在当前所计的非循环机的标识加上1），这一位就是β'的第n位。

可以看到，β'是完全可定义的，但是它是否可计算呢？

图灵没有给出任何终止机器的指令，他所面对的问题在现在称为终

止问题（halting problem，最早在马丁·戴维斯1958年的书《可计算性和不可解性》中提及[33]）。我们是否能够定义一个图灵机，使其能够确定另一个图灵机会停止还是永久运行下去？如果用循环的想法代替终止，仍存在同样的问题。一台图灵机能否分析另一台图灵机，并确定它最终的命运？

图灵一开始假设存在一台可以判断任意机器是否是非循环机器。在下面的讨论中，他使用机器的标准描述代替了描述数，这并不重要，二者总是可以转换的。

[247]

Let us suppose that there is such a process; that is to say, that we can invent a machine $\mathcal{D}$ which, when supplied with the S.D of any computing machine $\mathcal{M}$ will test this S.D and if $\mathcal{M}$ is circular will mark the S.D with the symbol “*u*” and if it is circle-free will mark it with “*s*”

我们假设存在这样一个过程，就是说，我们可以发明一台机器$\mathcal{D}$，当提供了任一机器$\mathcal{M}$的标准描述，它能够检测这个标准描述。如果$\mathcal{M}$是循环机，则用符号*u*标记这个标准描述；如果$\mathcal{M}$是非循环机，则用符号*s* 标记这个标准描述。

机器$\mathcal{D}$是决策机，符号*u*表示不符合要求（即循环机），*s*表示符合要求（非循环机）。图灵在第5节的结尾定义了这些术语（图灵论文第241页，本书第129页）。

By combining the machines $\mathcal{D}$ and $\mathcal{U}$ we could construct a machine $\mathcal{H}$ to compute the sequence β'.

结合机器$\mathcal{D}$和$\mathcal{U}$，我们可以构造机器$\mathcal{H}$来计算序列β'。

实际上，机器$\mathcal{H}$仍需要生成正整数，并将其转化成标准描述，而这些工作都很琐碎。对于机器$\mathcal{H}$生成的每一个正整数，$\mathcal{H}$用$\mathcal{D}$来决定这个数是否定义了一个符合要求的机器。如果是，则$\mathcal{H}$将标准描述传输给通用机$\mathcal{U}$来计算该序列。对第n个可计算序列，$\mathcal{U}$只需要运行机器至第n位。该位稍后将变成β'的第n位。因为机器$\mathcal{U}$受机器$\mathcal{H}$控制，当$\mathcal{H}$获得所需的特殊数位后便可终止$\mathcal{U}$。

对$\mathcal{H}$来说，有必要事先检查机器$\mathcal{D}$的标准描述，因为我们不希望$\mathcal{U}$由于运行了不符合要求机器而卡住。如果$\mathcal{H}$传递给$\mathcal{U}$一个不符合要求机器的标准描述，则不符合要求机器将无法打印任何数字位，整个进程将卡在这里而无法继续。

图灵实际上没有真正描述这个具有魔力的决策机$\mathscr{D}$会是什么样子，因此这可能存在一个巨大的暗示：不存在这种机器。下意识地想想，它至少是非常困难的。除了模拟目标机器并追踪它的每一步，机器$\mathscr{D}$如何能够确定一个机器是非循环的呢？

无论如何，机器$\mathscr{D}$和$\mathscr{U}$在处理标准描述（与描述数等价）上是相似的，都是在纸带上编码。

> The machine $\mathscr{D}$ may require a tape. We may suppose that it uses the *E*-squares beyond all symbols on *F*-squares, and that when it has reached its verdict all the rough work done by $\mathscr{D}$ is erased.

> 机器$\mathscr{D}$需要一条纸带。我们假设它使用了F-格上所有符号之外的E-格，并在最后得出结论时，抹去$\mathscr{D}$机器所做的中间工作。

只有符号*s*或*u*作为最终判断结果留了下来。

> The machine $\mathscr{H}$ has its motion divided into sections. In the first *N* - 1 sections, among other things, the integers 1, 2,…,*N* - 1 have been

written down and tested by the machine $\mathcal{D}$.

机器 $\mathcal{H}$ 的操作分成几个部分。在前*N*-1部分中，处理其他事情的同时，机器 $\mathcal{D}$ 写下整数1，2，…，*N*-1并加以检测。

这里“分成几个部分”并不意味着机器 $\mathcal{H}$ 存在不同的部分来处理不同的数。对应于每个数的不同的格局，都要求 $\mathcal{H}$ 是无限的。在一段时间内，图灵考虑了顺序性的操作。实际的过程必须对所有整数都是通用的：机器 $\mathcal{H}$ 一个接一个地产生正整数，并将其按顺序传递给 $\mathcal{D}$ 以决定它是否是符合要求的集合，如果是，用机器 $\mathcal{U}$ 来计算可计算序列中的某一些数位。

A certain number, say *R*(*N* - 1), of them have been found to be the D.N's of circle-free machines.

其中已发现有*R*(*N*-1)个数是非循环机的描述数。

*R*只表示已经被计数的非循环机。机器需要*R*来确定对于将要出现的非循环机需要计算多少位。

In the N-the section the machine tests the number N. If N is satisfactory, *i.e.*, if it is the D.N of a circle-free machine, then $R(N) = 1 + R(N - 1)$ and the first $R(N)$ figures of the sequence of which a D.N is N are calculated.

在第N部分中，机器检测数N。如果数N是可接受的，即它是非循环机的描述数，则$R(N)=1+R(N-1)$，并且将计算描述数为N的序列的前$R(N)$个数字。

举例来说，如果N是3 133 225 317，则$R(N)$为1（参见之前列出的关于前两个符合要求机器所对应的正整数）。到目前为止，只发现了一个符合要求机器。机器将确定N确实是非循环机的描述数，因此$R(N)$设为2，机器计算由3 133 225 317定义的机器的前两位，这两位都是1。用这些数位的第二个作为β'的第二位，依此类推即可！

The $R(N)$-th figure of this sequence is written down as one of the figures of the sequence *beta;'* computed by

通过H的计算，这个序列的第$R(N)$个数字将被写为序列β'的一位。

当然，一般情况是，数N根本不是机器或循环机的描述数。

If N is not satisfactory, then $R(N) = R(N - 1)$ and the machine goes on to the $(N + 1)$-th section of its motion.

如果N是不可接受的，则$R(N)=R(N-1)$，机器转向第N+1个部分继续运行。

要点在于，机器$\mathcal{H}$必须一个接一个地检测可能的描述数，并且对于每一个可接受的描述数，机器$\mathcal{H}$都必须运行至第$R(N)$位。图灵花费了很大力气来证明$\mathcal{H}$是非循环的。由初始假设，$\mathcal{D}$是非循环的，而$\mathcal{H}$只是简单地对每个可能的描述数运行$\mathcal{D}$。

From the construction of $\mathcal{H}$ we can see that $\mathcal{H}$ is circle-free. Each section of the motion of $\mathcal{H}$ comes to an end after a finite number of steps. For, by our assumption about $\mathcal{D}$, the decision as to whether N is satisfactory is reached in a finite number of steps. If N is not satisfactory, then the N-th section is finished. If N is satisfactory, this means that the machine $\mathcal{M}(N)$ whose D.N is N is circle-free, and therefore its $R(N)$-th figure can be calculated in a finite number of steps. When this figure has been calculated and written down as the $R(N)$-th figure of β', the N-th section is finished. Hence $\mathcal{H}$ is circle-free.

从$\mathcal{H}$的构成可以看出，$\mathcal{H}$是非循环的。$\mathcal{H}$的每一部分操作都在有限步内终止。由我们关于$\mathcal{D}$的假设，可在有限步内判定N是否是可接受的。如果N是不可接受的，那么第N部分终止；如果N是可接受的，意味着描述数是N的机器$\mathcal{M}$是非循环的，因此第$R(N)$个数字可在有限步内计算得到。当这个数字被计算出并写为β'的第$R(N)$个数字时，第N部分终止，所以$\mathcal{H}$是非循环的。

$\mathcal{H}$是一个图灵机，所以$\mathcal{H}$有一个描述数（图灵称之为K）。从某种程度讲，$\mathcal{H}$必须处理它自己的描述数，即$\mathcal{H}$需要确定它本身是否是非循环的。

Now let K be the D.N of $\mathcal{H}$. What does $\mathcal{H}$ do in the K-th section of its motion? It must test whether K is satisfactory, giving a verdict "*s*" or "*u*". Since K is the D.N of $\mathcal{H}$ and since $\mathcal{H}$ is circle-free, the verdict cannot be "*u*"

令K为$\mathcal{H}$的描述数，$\mathcal{H}$在第K部分的操作会是怎样的呢？它必须检测K是否可接受，给出一个判定符号s或u。因为K是$\mathcal{H}$

的描述数，而$\mathcal{H}$是非循环机，所以判定结果不可能是*u*。

图灵又说：

> On the other hand the verdict cannot be “*s*”.

另一方面，判定结果也不可能是*s*。

根本的问题在于，$\mathcal{H}$陷入了无限循环中。在$\mathcal{H}$计数到数字*K*（它自身的描述数）之前，$\mathcal{H}$已经分析了1到*K*-1的所有正整数。非循环机的数目在此时为*R*(*K*-1)，并且β'的前*R*(*K*-1)位已经计算出来了。

β'的第*R*(*K*)位是什么呢？为了得到这一位，$\mathcal{H}$必须追踪它自己的操作，这意味着它不得不复制它计数到*K*之前的所有操作，然后再重新开始这个过程。这就是$\mathcal{H}$不可能是非循环的原因。

> For if it were, then in the *K*-th section of its motion $\mathcal{H}$ would be bound to compute the first *R*(*K* - 1) + 1 = *R*(*K*) figures of the sequence computed by the machine with *K* as its D.N and to write down the *R*(*K*)-th as a figure of the sequence computed by $\mathcal{H}$. The computation of the first *R*(*K*) - 1 figures would be carried out all right, but the instructions for calculating the *R*(*K*)-th would amount to “calculate the

first $R(K)$ figures computed by H and write down the $R(K)$-th”. This $R(K)$-th figure would never be found.

因为如果是s，那么机器$\mathcal{H}$的第K部分操作将一定要计算以K为描述数的机器所产生序列的前$R(K-1)+1=R(K)$个数字，并将第$R(K)$个数写下来作为$\mathcal{H}$计算的序列的数字。前$R(K-1)$个数字的计算不会有问题，但是计算第$R(K)$个数字的指令相当于“计算由H计算的前$R(K)$个数字，并写下第$R(K)$个数字”，这样的第$R(K)$个数字永远不会找到。

（倒数第二行的H应该是$\mathcal{H}$。）

基于其他机器产生的序列，$\mathcal{H}$生成了一列数位。这很直接，我们可以想象成这是一个正在产生1/3、π、2的平方根的二进制序列的机器，但是$\mathcal{H}$如何获得这个生成的序列里的第K位呢？这需要它从自身获得这一位，但是这不可能，因为$\mathcal{H}$只能从其他机器获得数位。

所以，当$\mathcal{H}$遇到它的描述数时有了点麻烦。就不能忽略它吗？可以，但是我们会发现，每个可计算序列都可以用不同的机器来计算。这些机器用不同的方式计算相同的序列，或者它们还有一些不必要的指令。$\mathcal{H}$可能也需要跳过这些类似的机器。不能计算β'但能计算接近β'的数，比如交换了β'的第27位和第54位，这样的机器又如何？这样的机

器有无数个，必须防止它们给$\mathcal{H}$带来负担——一个无法完成的负担。

I.e., $\mathcal{H}$ is circular, contrary both to what we have found in the last paragraph and to the verdict *“s”*. Thus both verdicts are impossible and we conclude that there can be no machine $\mathcal{D}$.

换言之，$\mathcal{H}$是循环的，这和我们在上一段的结论以及断言*s*都是相反的。因此这些断言都是不可能的，我们可以总结出没有机器$\mathcal{D}$。

没有通用过程来判断一台机器是否是非循环的。这就是说，没有这样的计算机程序，可以决定其他计算机程序的最终命运。

图灵也解决了对角线法的矛盾。他首先指出可计算数是可数的，而对角线法似乎表明你可以创造一个不在列表中的可计算数。图灵证明了对角线法是无法通过有限步计算的，因此它不可计算。可计算数可能是可数的，但它们实际上不能在有限步内被枚举出来。

图灵没有这样结束这一节。他假设了一个机器$\mathcal{E}$，可能表示“过去的输出”。

[248]

We can show further that *there can be no machine* $\mathcal{E}$ *which,*

when supplied with the S.D of an aribtrary machine $\mathcal{M}$*, will determine whether* $\mathcal{M}$ *ever prints a given symbol* (0 *say*).

我们可以进一步证明，没有这样的机器 $\mathcal{E}$ ，当给它提供了任意一台机器 $\mathcal{M}$ 的标准描述时，它可以判断 $\mathcal{M}$ 是否曾经打印过给定的符号（比如0）。

在论文的最后一节，图灵需要机器 $\mathcal{E}$ 来证明判定性问题是无解的。这里，他将证明机器 $\mathcal{E}$ 是不存在的：首先如果机器 $\mathcal{E}$ 存在，那么就存在一个过程来判断机器是否经常无限次打印0，但这又意味着存在一个类似的过程来判断机器是否经常无限次打印1。如果你有能力来判断机器是经常无限次打印0还是经常无限次打印1（或者都打印），那么你就有能力判断机器是否是非循环的。这样的过程已经被证明是不存在的，所以机器 $\mathcal{E}$ 也一定不存在。

We will first show that, if there is a machine $\mathcal{E}$, then there is a general process for determining whether a given machine $\mathcal{M}$ prints 0 infinitely often.

我们首先证明，如果存在这样的机器$\mathcal{E}$，就会有一个通用过程来判定机器$\mathcal{M}$是否经常无限次打印0。

图灵通过定义任意机器$\mathcal{M}$的变体这一古怪的方法，来对此进行判断。

Let $\mathcal{M}_1$ be a machine which prints the same sequence as $\mathcal{M}$, except that in the position where the first 0 printed by $\mathcal{M}$ stands, $\mathcal{M}_1$ prints $\bar{0}$. $\mathcal{M}_2$ is to have the first two symbols 0 replaced by $\bar{0}$, and so on. Thus, if $\mathcal{M}$ were to print

$$ABA01AAB0010AB\ldots,$$

then $\mathcal{M}_1$ would print

$$ABA\bar{0}1AAB0010AB\ldots$$

and $\mathcal{M}_2$ would print

$$ABA\bar{0}1AAB\bar{0}010AB\ldots.$$

假设机器$\mathcal{M}_1$打印的序列与机器$\mathcal{M}$打印的序列相同，所不同的只是，在$\mathcal{M}$打印第一个0的地方，$\mathcal{M}_1$打印$\overline{0}$；$\mathcal{M}_2$将头两个符号0替换成$\overline{0}$，以此类推。这样，如果$\mathcal{M}$打印

*ABA*01*AAB*0010*AB*…，

那么$\mathcal{M}_1$会打印

$$ABA\overline{0}1AAB0010AB\ldots,$$

$\mathcal{M}_2$会打印

$$ABA\overline{0}1AAB\overline{0}010AB\ldots,$$

如果你有一台机器$\mathcal{M}$，就能定义一台机器读取$\mathcal{M}$的标准描述，然后计算$\mathcal{M}_1$、$\mathcal{M}_2$等的标准描述吗？图灵说可以，他称这样的机器为$\mathcal{F}$。

Now let $\mathcal{F}$ be a machine which, when supplied with the S.D of $\mathcal{M}$, will write down successively the S.D of $\mathcal{M}$, of $\mathcal{M}_1$, of

$\mathcal{M}_2$ … (there is such a machine).

假设有一台机器 $\mathcal{F}$ ，当给它提供了 $\mathcal{M}$ 的标准描述后，它就可以连续写出 $\mathcal{M}$ 的标准描述， $\mathcal{M}_1$ 的标准描述， $\mathcal{M}_2$ 的标准描述……（这样的机器是存在的。）

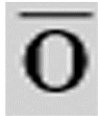
为了让我们自己相信 $\mathcal{F}$ 似乎是可信的，我们考虑一台非常简单的机器，它选择性地打印0和1，即1/3的二进制格式但不跳过任何空间：

q_1 None $P0$, R q_2
q_2 None $P1$, R q_1

这是机器 $\mathcal{M}$ 。下面是机器 $\mathcal{M}_1$：

q_1	None	$P\bar{0}, R$	q_4
q_2	None	$P1, R$	q_1
q_3	None	$P0, R$	q_4
q_4	None	$P1, R$	q_3

所有的初始格局（包括 $\mathcal{M}$ 和 $\mathcal{M}_1$ ）都只是简单的重复，然后给出不同的m-格局。在第一组格局中，所有应该打印0的地方现在打印

，然后它会跳到第二组合适的格局中。$\mathcal{M}_2$有三组：

q_1	None	$P\overline{0}, R$	q_4
q_2	None	$P1, R$	q_1
q_3	None	$P\overline{0}, R$	q_6
q_4	None	$P1, R$	q_3
q_5	None	$P0, R$	q_6
q_6	None	$P1, R$	q_5

你可能注意到，这些修改过的机器永远不会遇到格局q_2，但是对于这台机器，这是偶然的。

因此，$\mathcal{F}$存在是完全可信的。注意这些$\mathcal{M}$机器之间的关系：如果$\mathcal{M}$从来不打印0，那么$\mathcal{M}_1$、$\mathcal{M}_2$等也不会。如果$\mathcal{M}$只打印一次0，那么$\mathcal{M}_1$、$\mathcal{M}_2$等永远不会打印0。如果$\mathcal{M}$打印了0两次，那么$\mathcal{M}_1$打印0一次，而$\mathcal{M}_2$等永远不会打印0。如果$\mathcal{M}$经常无限次打印0，那么$\mathcal{M}_1$、$\mathcal{M}_2$等也会如此。

现在回忆一下机器$\mathcal{E}$，它被假设可以判断一个机器是否打印过0。

We combine $\mathcal{F}$ with $\mathcal{E}$ and obtain a new machine, $\mathcal{G}$. In the motion of $\mathcal{G}$ first $\mathcal{F}$ is used to write down the S.D of $\mathcal{M}$, and then $\mathcal{E}$ tests its, :0: is written if it is found that $\mathcal{M}$ never prints 0; then $\mathcal{F}$ writes the S.D of $\mathcal{M}_1$, and this is tested, :0: being printed if and only if $\mathcal{M}_1$ never prints 0, and so on.

我们将机器$\mathcal{E}$和机器$\mathcal{F}$合并为一个新的机器$\mathcal{G}$。$\mathcal{G}$的第一个动作就是使用$\mathcal{F}$写下$\mathcal{M}$的标准描述，然后让$\mathcal{E}$来测试它。如果$\mathcal{M}$从未打印0，那么写下:0:。然后$\mathcal{F}$写下$\mathcal{M}_1$的标准描述，再进行测试。仅当$\mathcal{M}_1$从未打印0时才写下:0:。如此一直继续下去。

$\mathcal{G}$使用$\mathcal{F}$来生成$\mathcal{M}$、$\mathcal{M}_1$、$\mathcal{M}_2$等的描述数，然后$\mathcal{E}$来判断产生结果的机器是否打印过0。如果它从未打印过0，那么$\mathcal{G}$打印0。

结果是这样的：如果$\mathcal{M}$从未打印过0或仅有限次打印0，那么$\mathcal{G}$经常无限次打印0。如果$\mathcal{M}$经常无限次打印0，则$\mathcal{G}$不打印0。

Now let us test $\mathcal{G}$ with $\mathcal{E}$. If it is found that $\mathcal{G}$ never prints 0, then $\mathcal{M}$ prints 0 infinitely often; if $\mathcal{G}$ prints 0 sometimes, then $\mathcal{M}$ does not print 0 infinitely often.

现在，我们用$\mathcal{E}$来测试$\mathcal{G}$。如果发现$\mathcal{G}$从未打印过0，那么$\mathcal{M}$经常无限次打印0；如果$\mathcal{G}$有时打印0，那么$\mathcal{M}$不经常无限次打印0。

这意味着，$\mathcal{G}$可以告诉我们$\mathcal{M}$是否经常无限次打印0。这是通过它自己不打印0来告诉我们的。

Similarly there is a general process for determining whether $\mathcal{M}$ prints 1 infinitely often. By a combination of these processes we have a

process for determining whether $\mathcal{M}$ prints an infinity of figures, *i.e.* we have a process for determining whether $\mathcal{M}$ is circle-free. There can therefore be no machine $\mathcal{E}$.

类似地，也有一个方法来判断$\mathcal{M}$是否经常无限次打印1。通过将这些过程结合起来，我们有了一个过程来判断$\mathcal{M}$是否无限次打印一个数字。也就是，我们可以有一个过程来判断$\mathcal{M}$是否是非循环的。所以不会存在这样的机器$\mathcal{E}$。

图灵也给出了另一个反证法来证明$\mathcal{E}$并不存在。因为它最终会导致$\mathcal{D}$存在——机器$\mathcal{D}$可以判断其他机器是否是非循环的，而事实上，这种机器不存在。

图灵在结束这一节时提醒道，我们真的需要检查人类计算者和图灵机是等价的这一假设，因为我们在很大程度上依赖于此。

The expression "there is a general process for determining …" has been used throughout this section as equivalent to "there is a machine which will determine …". This usage can be justified if and only if we can justify our definition of "computable".

本节中使用的“有一个通用过程来判断……”的表述。等价于“有这样一台机器能判断……”。这种用法可以证明是对的，当且仅当我们可以证明“可计算”的定义。

这个证明将在下一节讨论。

图灵然后略微提及了这个证明的另一个方面，这些内容将在本书第三部分介绍。图灵将图灵机的输出释义为“可计算数”，而图灵机在这方面具有更多的灵活性。例如输出如下序列的机器：

0011010100010100010100010000010…

这看起来像是一个数，实际上，这是一个“素数”机器的输出，我们将这样的数记为IsPrime(*n*)。对于该序列的第*n*个数字（*n*从0开始），如果*n*是素数，则IsPrime(*n*)的值为1，否则IsPrime(*n*)的值为0。这台机器输出的序列表明了2、3、5、7、11、13、17、19、23和29都是素数。这样的机器似乎完全可信，但是它并不是真正对某个数字进行计算，而是告诉我们一些自然数的信息。

For each of these “general process” problems can be expressed as a problem concerning a general process for determining whether a given integer *n* has a property *G*(*n*) [*e.g.* *G*(*n*) might mean “*n* is satisfactory” or “*n* is the Gödel representation of a provable formula”], and this is equivalent to computing a number whose *n*-th figure is 1 if *G* (*n*) is true and 0 if it is false.

每一个这样的“通用过程”问题都能表示成，判定给定整数*n*是否具有性质*G*(*n*)（如*G*(*n*)表示“*n*是可接受的”或者“*n*是可证明公式的哥德尔表示”）的通用过程，并且等价于计算一个数，如果*G*(*n*)成立，则这个数的第*n*个数字为1，否则为0。

现在，图灵似乎将他的计算机器和数理逻辑联系在了一起，这种联系本书在第三部分提及。符号0和1不仅仅可以是二进制位，同时还可用来表示真或假，就像乔治·布尔多年前认识到的那样。

考虑一组关于自然数的函数，它们会返回真或假（1或0）。

IsPrime(*n*)

IsEven(*n*)

IsOdd(*n*)

IsLessThanTen(n)

IsMultipleOfTwentyTwo(n)

它们有时称为布尔函数，可以通过由图灵机对n=0, 1, 2, 3…输出0和1的序列来实现，其中判定奇数的函数IsOdd输出的序列与图灵第一台样机输出的序列交替相同。

图灵已经说明了这些可计算序列是可数的，因此和它们的名字一样，它们可以按字母排序，例如在康托尔关于超限数的记号中，自然数的所有可计算、可按字母排序的布尔函数集的势为$\aleph_0$，和可数集的势相同。

每个布尔函数都会针对自然数的子集返回真或者1。例如，IsPrime针对下列的自然数集合返回1：

{2, 3, 5, 7,11,13,…}

每个这样的布尔函数都和不同的自然数集合相关联。可以回想一下第2章，所有子集的集合叫做超集，如果原始集合的势是$\aleph_0$，那么这个超集的势就是$2^{\aleph_0}$。

所有可想到的布尔函数的集合的势是$2^{\aleph_0}$，而所有可计算布尔函数（其实就是所有可以用英语名字描述的布尔函数的集合）的势是$\aleph_0$。这是可想到的和可计算的之间的另一个重大鸿沟。

---

[1] Paul Ceruzzi，*Reckoners: The Prehistory of the Digital Computer, from Relays to the Stored Program Concept, 1935-1945*（Greenwood Press，1983），43。 Ceruzzi的引用源于Howard Aiken，“The Future of Automatic Computing Machinery”，*Elektronische Rechenanlage und Informationsverarbeitung*（Darmstadt，1956），33。

[2] 阿兰·图灵，“Computing Machinery and Intelligence”，*Mind*，Vol. LIX，No. 236（October 1950），441- 2。

[3] Martin Campbell-Kelly and William Aspray，*Computer: A History of the Information Machine*（Basic Books，1996）。

[4] 马丁·戴维斯，*The Universal Computer: The Road from Leibniz to Turing*（Norton，2000）。

[5] 安德鲁·霍奇斯，*Alan Turing: The Enigma*（Simon & Schuster，

1983），p. 95。
[6] 霍奇斯，*Alan Turing*，118。
[7] 霍奇斯，*Alan Turing*，124。
[8] 霍奇斯，*Alan Turing*，145。
[9] Nancy Stern，“John von Neumann’s Influence on Electronic Digital Computing, 1944-1946”，*Annals of the History of Computing*，Vol. 2，No. 4（October 1980），353。
[10] 重印于Brian Randell，ed.，*The Origins of Digital Computers*（Springer，1973）。
[11] W. S. MacCulloch and W. Pitts，“A Logical Calculus of the Ideas Immanent in Nervous Activity”，*Bulletin of Mathematical Biophysics*，Vol. 5（1943），115-133。
[12] 此信引用于B. Jack Copeland，ed.，*The Essential Turing: The Ideas that Gave Birth to the Computer Age*（Oxford University Press，2004），22。这封信是6页“Turing, von Neumann, and the Computer”的一部分，详见Copeland对“Computable Numbers”论文的指导。
[13] B. Jack Copeland and Diane Proudfoot，“Turing and the Computer” in B. Jack Copeland，ed.，*Alan Turing’s Automatic Computing Engine*: *The Master Codebreaker’s Struggle to Build the Modern Computer*（Oxford University Press，2005），116
[14] 本书作者的*Code: The Hidden Language of Computer Hardware and Software*（Microsoft Press，1999），244
[15] 霍奇斯，*Alan Turing*，268。
[16] 霍奇斯，*Alan Turing*，554（注释5.7）。
[17] 全名查尔斯·高尔顿·达尔文，其祖父是查尔斯·罗伯特·达尔文，英国生物学家，进化论的奠基人。——编者注
[18] B. E. Carpenter and R. W. Doran，*A.M. Turing’s ACE Report of 1946 and Other Papers*（MIT Press，1986）的介绍，5-6。
[19] *ACE Report*，21。
[20] *ACE Report*，116。
[21] *ACE Report*，76。
[22] *ACE Report*，112-113。
[23] *ACE Report*，112。
[24] *ACE Report*，113-114。
[25] *ACE Report*，123。
[26] Martin Campbell-Kelly，“Programming the Mark I: Early Programming Activity at the University of Manchester”，*Annals of the*

History of Computing，Vol. 2，No. 2（April 1980），134。
[27] Campbell-Kelly，“Programming the Mark I”，147。手册也可以在这里找到：www.
alanturing.net/turing_archive/archive/index/manchesterindex.html。
[28] rand.c © 1985-1997，Microsoft Corporation。为了更清晰，rand函数的细节做了调整。
[29] 在本书作者的书*Code*中可以找到关于向上溢出和向下溢出的讨论，153-154。
[30] 约翰·冯·诺依曼，*Collected Works, Volume V, Design of Computer, Theory of Automata, and Numerical Analysis*（Macmillan，1963），768。这个评论首先出现在1949年的Monte Carlo方法讨论会中。
[31] Martin Campbell-Kelly，“Programming the Mark I”，136。
[32] 由剑桥大学数学教授E. W. 霍布森（1856—1933）所写的影响深远的书，全名是 *The Theory of Functions of a Real Variable and the Theory of Fourier's Series*，第一版由剑桥大学出版社于1907出版，图灵所引用的第二版要追溯到 1921，因为很多材料要加到其中，所以卷2到1926年才出版。卷2也称为第二版。卷1在1927年修订，称为第三版。卷1的第三版和卷2的第二版分别由Harren Press于1950年和Dover Books于1957年重新出版，这些可能是能追述到的最简单的版本。图灵引用的讨论在卷1第三版的第84页和第85页。接后的就是康托尔的对角证明。
[33] 马丁·戴维斯，*Computability and Unsolvability*（McGraw-Hill，1958），70。戴维斯认为是他在1952的演讲中首次使用了此词。（参见Copeland的*The Essential Turing*，40，脚注61。）这个概念也出现在Stephen Cole Kleene的*Introduction to Metamathematics*（Van Nostrand，1952）的第13章。

第11章 机器与人

阿兰·图灵在其论文第1节的开头（本书第59页）定义可计算数时写道：“在§9之前，我们不会真正尝试证明这个定义的合理性”。现在，我们讲到第9节了，图灵传记作者安德鲁·霍奇斯将论文的剩余部分誉为“有史以来数学类论文中最不寻常的部分”。[1]

图灵试图论证，图灵机的计算能力等同于一部执行明确定义了数学过程的人类计算者。因此，如果算法过程是图灵机不可解的，那么它对于人类而言也是不可解的。这个想法后来得名图灵论题，也称为邱奇-图灵论题。称其为“论题”是因为它是个过于模糊的概念，不能接受严格的数学证明。这个论题还被扩展到了其他数字计算机上，这些计算机的计算能力都没有图灵机强。

本章只会讨论第9节的第一部分，阅读第9节的其余内容需要数学逻辑知识，本书的第三部分再对其进行阐述。对于大多数内容，我不会打断图灵本人的分析。以下是马丁·戴维斯的总结。

> 图灵的“分析过程”是应用哲学的一个精彩展现。在这个分析过程里，图灵以人类如何进行计算开始，摒弃了无关的细节，简明扼要地得到分析结果，这个结果就是大家熟悉的运行在单向无限长线性纸带上的有限状态机模型。[2]

[249]

9. The extent of the computable numbers.

No attempt has yet been made to show that the “computable” numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is “What are the possible processes which can be carried out in computing a

number?”

The arguments which I shall use are of three kinds.

(*a*) A direct appeal to intuition.

(*b*) A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).

(*c*) Giving examples of large classes of numbers which are computable.

9. 可计算数的范畴

现在还未证明，“可计算”数包括所有被自然当做可以计算的数字。所有能给出的论据本质上都局限在直觉上，没有令人满意的数学说服力。真正有争议的问题是：“对数进行计算的可能的过程有哪些？”

我能给出的论据有三类：

(a) 直觉的引导；

(b) 关于两种定义等价的证明（新的定义有更多的直觉成分）；

(c) 给出大量可计算数的示例；

(b) 论据将在本书第三部分讨论，(c)论据在图灵论文的第10节有进一步阐述。

Once it is granted that computable numbers are all “computable”, several other propositions of the same character follow. In particular, it follows that, if there is a general process for determining whether a formula of the Hilbert function calculus is provable, then the determination can be carried out by a machine.

一旦承认所有的可计算数都是“可以计算的”，便会产生具有相同特征的另外一些命题。特别是可以得出，如果存在可以判定希尔伯特函数演算的方程是否可证明的通用过程，那么这个判定就可以用机器进行计算。

“希尔伯特函数演算”就是今天数学逻辑体系里的“一阶谓词逻辑”。希尔伯特在这种逻辑下定义判定性问题。图灵不太可能知道“用机器进行计算”的过程就是海因里希·贝曼之前谈及判定性问题时提到的观点（第40页）。贝曼的论述直到最近才发表。

I. [Type (*a*)]. This argument is only an elaboration of the ideas of § 1.

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent[†]. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as

[250]

17 or 999999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of symbols). The differences from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.

[†] If we regard a symbol as literally printed on a square we may suppose that the square is $0 \leq x \leq 1$, $0 \leq y \leq 1$. The symbol is defined as a set of points in this square, viz. the set occupied by printer's ink. If these sets are restricted to be measurable, we can define the “distance” between two symbols as the cost of transforming one symbol into the other if the cost of moving unit area of printer's ink unit distance is

unity, and there is an infinite supply of ink at $x = 2$, $y = 0$. With this topology the symbols form a conditionally compact space.

I. [类型(a)]。这个论据只是对§1观点的详细阐述。

计算通常可以通过在纸上书写某些符号来完成。我们可以假设这张纸就像小孩子的算术书，分成一个个方格。在初等算术中，有时会利用纸的二维性。但是，这种做法总是可回避的，并且我认为，大家应该认同纸的二维特性对于计算并不重要。我假定计算是在一张一维的纸上完成的，例如在一条分成方格的纸带上。另外，假设可打印符号的数目是有限的。如果我们允许数目是无限的，那么将会存在一些差异程度任意小的符号。†限制符号数目并不会有严重的影响，因为总是可以使用符号序列代替单个符号。通过这种方法，像17或999999999999999这样的阿拉伯数字将被认为是单个符号。类似地，任何欧洲语言里的单词都被当做单个符号（但是，汉语倾向于拥有可枚举的无限的符号）。在我们的观点里，单一符号和组合符号的区别在于，太长的组合符号不能一眼就识别出来。这是与我们的经验相一致的。我们不能一下子辨别出9999999999999999与999999999999999是否是同一个数。

---

† 如果我们认为一个符号是打印在一个$0 \le x \le 1$，$0 \le y \le 1$的方格内，那么可以将这个符号定义为这个方格内一系列点组成的集，即被打印点墨占据的点集。如果这些点集是可测量的，且将单位面积的打印点墨移动单位长度的开销是统一的，我们可以将两个符号的“距离”定义为将一个符号变换为另一个符号的开销，而在$x = 2$，$y = 0$的地方有无限量的点墨。在这种拓扑结构下，符号有条件地组成了一个紧凑的区域。

下一句话里，图灵谈到了“计算机”，当然，他指的是人类“计算者”。

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his “state of mind” at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also

suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be “arbitrarily close” and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

计算者任一时刻的行为都由彼时他观察到的符号和彼时他的“思维状态”决定。我们可以假设，对于符号的数目或者计算者在某一个时刻所能观察到的方格，存在一个上限B。如果他想观察到更多，就必须继续观察。我们也假定，需要考虑的思维状态的数量是有限的。这样做的理由和限制符号数目的理由是相同的。如果我们允许思维状态是无限的，那么它们之中有些状态将会因“无限地接近”而造成混淆。同样地，这种限制不会对计算造成很大的影响，因为避免使用更复杂的思维状态时可以通过在纸带上写上更多的符号来实现。

1972年，库尔特·哥德尔针对这节中图灵的分析写了短评，称其为“这是图灵分析中的一个哲学错误”。[3]哥德尔认为，“图灵分析里所指的思维并不是静止的，而是持续变化的”，而且精神上的思维状态可能会更加趋于无穷。这种分歧代表了对“思维最终是来自于大脑的机械过程”持正反观点的两派之间的基本冲突。

Let us imagine the operations performed by the computer to be split up into “simple operations” which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape. We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer. We may suppose that in a simple operation not more than one symbol is altered. Any other changes can be split up into simple changes of this kind. The situation in regard to the squares whose symbols may be altered in this way is the same as in regard to the observed squares. We may, therefore,

without loss of generality, assume that the squares whose symbols are changed are always “observed” squares.

Besides these changes of symbols, the simple operations must include changes of distribution of observed squares. The new observed squares must be immediately recognisable by the computer. I think it is reasonable to suppose that they can only be squares whose distance from the closest of the immediately previously observed squares does not exceed a certain fixed amount. Let us say that each of the new observed squares is within *L* squares of an immediately previously observed square.

In connection with “immediate recognisability”, it may be thought that there are other kinds of square which are immediately recognisable. In particular, squares marked by special symbols might be taken as imme-

[251]

diately recognisable. Now if these squares are marked only by single symbols there can be only a finite number of them, and we should not upset our theory by adjoining these marked squares to the observed squares. If, on the other hand, they are marked by a sequence of symbols, we cannot regard the process of recognition as a simple process. This is a fundamental point and should be illustrated. In most mathematical papers the equations and theorems are numbered. Normally the numbers do not go beyond (say) 1000. It is, therefore, possible to recognise a theorem at a glance by its number. But if the paper was very long, we might reach Theorem 157767733443477; then, further on in the paper, we might find “… hence (applying Theorem 157767733443477) we have …”. In order to make sure which was the relevant theorem we should have to compare the two numbers figure by figure, possibly ticking the figures off in pencil to make sure of their not being counted twice. If in spite of this it is still thought that there are other “immediately recognisable” squares, it does not upset my contention so long as these squares can be found by some process of which my type of machine is capable. This idea is developed in III below.

我们想象一下，把机器的操作分解成“简单操作”，即最基本的操作，以至于不能想象它们能够再分解。每个这样的操作都是由计算者及其纸带组成的物理系统的变化构成的。如果我们知道纸带上的符号序列，就知道了系统的状态，这些都是由计算者（通过特定次序）和计算者的思维状态观察到的。我们假设在一个简单操作里最多有一个符号会改变。所有其他的变化都可以分解成这种简单的变化，其中，符号会被改变的那些方格的情况与被观察到的方格相同。因此，我们可以不失一般性地假设，符号变化的方格总是那些“被观察”的方格。

除了这些符号上的改变，简单操作还必须包含被观察方格分布的变化。新的被观察方格必须可以立即被计算者识别。我想可以合理地假设这些方格满足它们与最近的前一个立即被观察方格的距离不超过某个特定值。我们假设每个新的被观察方格距离前一个立即被观察方格的距离不超过*L*个方格。

谈到“立即识别”，也许存在其他类型的被立即识别方格。特别地，用特殊符号标记的方格可能被认为是立即识别的。若这些方格是由单个符号标记的，那么这样的方格数目就是有限的，可以遵循我们的理论将这些被标记的方格毗连到被观察的方格边。另一方面，如果标记方格的是序列符号，我们就不能将识别过程当做是一个简单过程。这是一个应该强调的基本出发点。在绝大多数数学论文里，方程和定理都是附上标号的。通常，这些标号不会超过1000，因此扫一眼标号就能迅速识别一个定理。但是如果论文很长，标记的定理可能到157767733443477号，接着我们可能要表述为“……因此通过应用157767733443477号定理，我们得到……”。为了确定相关定理，我们需要一位一位地比较两个数，可能需要用铅笔将比较过的数字划掉以免重复算两次。如果除了这些还存在其他“立即识别”的方格，只要这些方格可以在我的机器的运行过程中找到，也不会与我的论点相违背。这个观点会在第III部分进一步阐述。

当图灵说“用铅笔将比较过的数字划掉”时，他很可能指的是用非数字符号“标记”方格这样类似的机器操作。

The simple operations must therefore include:
(*a*) Changes of the symbol on one of the observed squares.
(*b*) Changes of one of the squares observed to another square

within *L* squares of one of the previously observed squares.

It may be that some of these changes necessarily involve a change of state of mind. The most general single operation must therefore be taken to be one of the following:

(*A*) A possible change (*a*) of symbol together with a possible change of state of mind.

(*B*) A possible change (*b*) of observed squares, together with a possible change of state of mind.

The operation actually performed is determined, as has been suggested on p. 250, by the state of mind of the computer and the observed symbols. In particular, they determine the state of mind of the computer after the operation is carried out.

因此，简单操作必须包括：

(a) 改变一个被观察方格的符号。

(b) 将一个被观察方格移动到与前一个被观察方格距离*L*以内的位置上。

这些改变有可能涉及一系列思维状态的转变。因此，最普遍意义上的单个操作必须是下列情况之一：

(A) 一个可能的(a)型的符号改变以及一个潜在的思维状态转变。

(B) 一个可能的(b)型的被观察方格的改变以及一个潜在的思维状态转变。

第250页已经说过，操作实际上是由计算者的思维状态和被观察的符号所决定的。特别是，操作执行后，计算者的思维状态就确定了。

这些只是他对论文前一页的引用。

We may now construct a machine to do the work of this computer. To each state of mind of the computer corresponds an "*m*-configuration" of the machine. The machine scans *B* squares corresponding to the *B* squares observed by the computer. In any move the machine can change a symbol on a scanned square or can change any one of the scanned squares to another square distant not more than *L* squares from one of

the other scanned

[252]

squares. The move which is done, and the succeeding configuration, are determined by the scanned symbol and the *m*-configuration. The machines just described do not differ very essentially from computing machines as defined in § 2, and corresponding to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say the sequence computed by the computer.

我们现在可以构造一台做这种计算者工作的机器了。对于计算者的任意一个思维状态，机器都有相应的一个m-格局。对应计算者观察*B*个方格，机器扫描*B*个方格。对于任意一次移动，机器要么改变被扫描方格上的符号，要么将任意一个被扫描方格移动到距离其他被扫描方格距离不超过*L*的位置。当完成移动后，后续的格局都由扫描符和m-格局决定。这里描述的机器与§2定义的计算机器并没有本质的区别，对应于任意这一类机器，都可以构造一个计算机器去计算相同的序列，也就是由计算者计算的序列。

这就是，人类计算者。

我们暂且在这里打住。图灵在本节的第二个论据从对“受限希尔伯特函数演算”的引用开始，接着对这个演算进行了陈述，为开始阅读本书第三部分提供一些背景知识。

图灵对人脑和机器之间联系的着迷，在1936年发表可计算数论文之后仍延续了很久。图灵的另一篇著名论文“计算机械与人工智能”发表在1950年10月的哲学期刊*Mind*上。

“机器能思考吗?”图灵问。他发明了一个测试，这个测试需要一个人坐在电传打字机前（在现代，类似于短消息，或者其他允许人们在看不见也听不见对方的情况下相互通信的手段）。这个人问问题，接受答案。如果另一端是计算机，而这个人无法判断它是否是一台计算机，那么就说计算机是具备人类智能的。

这就是著名的图灵测试，它至今依旧存在争议。任何对图灵测试有适当反对意见的人都应该读一读图灵的论文，里面有对很多合情合理的反对意见的解答。

图灵喜欢用术语“智能”而不是“思考”来处理这个问题，因为“思考”暗含在计算机内部进行的特定活动。

> “机器能思考吗?”我认为，这个原始问题过于无意义，不值得讨论。不过，我认为到这个世纪末，这样的说法以及一般的教育观点都会有很大改观，那时候再谈及机器思考将不会受到抵触和反对。[4]

上个世纪末已经过了，若说有什么改观，那就是比以往任何时候都多的人知道了计算机能够干什么，但那不是“思考”。我们还未到可以期待计算机媲美人类智能的地步，并且在通常情况下，当我们认为计算机应用能以一种完全确定的形式运行时才能很好地运用它进行工作。一个尝试做任何“智能”工作的计算机程序通常就像个2岁的孩子，盯着一幅新画的蜡笔壁画，然后诉求“我想你会喜欢它的”。

在科幻小说的世界里，图灵预言成真，就像下面描述的一台有史以来最著名的虚构计算机：

> 哈尔能否思考的问题已经被英国数学家阿兰·图灵在20世纪40年代解决了。图灵指出，如果一个人能够与一台机器进行一次足够长的对话，无论是通过电传打字机还是麦克风都不重要，假如不能分辨回答是来自一台机器或者一个人，那么从感性的定义上，这台机器就是能思考的。哈尔能够轻易通过图灵测试。[5]

如果图灵能够活到56岁，也就是1968年《2001：太空漫游》的小说和电影发行的时候，他也许会感到十分有趣，因为计算机已经相当智能，能够体验到精神崩溃。

1950年夏天，图灵搬到了位于曼彻斯特以南10英里的威姆斯洛。他对形态形成学产生了兴趣，这是一门研究组织细胞如何发展和分化，形成各种各样模式和形态的物种学科。这个研究涉及在曼彻斯特的计算机上运行仿真程序。

1951年3月15日，阿兰·图灵因其在可计算数方面所做的工作，成为英国皇家学会的会士，举荐他的是麦克斯·纽曼和伯特兰·罗素。那天晚上，BBC播放了图灵题为“数字计算机能够思考吗？”的录音谈话（这个广播的录音和其他图灵所有讲话的录音都已不知所踪了）。

1951年12月，接连发生的一系列事件对日后产生了很大的影响。图灵在曼彻斯特的街上遇见了一个年轻人阿诺德·穆雷。工人出身的穆雷正处于偷窃罪缓刑期，也没有工作。图灵和穆雷共进了午餐，一起回到了图灵的家里。在接下来的一个月，他们还相会了几次。

1952年1月底，图灵发现住所遭窃。他报了警，警察检查了现场的指纹。图灵指控阿诺德·穆雷行窃，而穆雷声称自己是无辜的，并指认

真正的罪犯是自己的一个旧相识哈里。警方也在图灵的住所找到了哈里的指纹。哈里彼时因为其他一些事情正在坐牢，在被问到图灵一案时，哈里告诉了警方图灵和其朋友间一些很私密的情况。

1952年2月7日，就在乔治六世驾崩，他的长女伊丽莎白继位的隔日，警方传讯了阿兰·图灵。在几轮审讯后，图灵承认了与穆雷之间的关系。这个供认让图灵遭受了牢狱之灾，因为根据1885年的刑法修正案第11节：

> 任何男性，公开或私下，组织或参与组织，引诱或试图引诱其他男性进行严重猥亵的行为，都应该视为不法行为，并理应被法庭判处不超过2年、可带劳役或不带劳役的监禁。[6]

法律并没有定义“严重猥亵”的概念，但通常指的是诸如手淫和口交之类的行为。其他法律还就更严重的肛交行为（即“鸡奸”，英国司法系统里用了这个术语）制定了刑律。

恶名昭著的刑法修正案第11节从一开始就颇受非议。1885年的刑法修正案自称是“一部进一步保护妇女和未成年少女，抑制妓院和其他恶习的法律”。这部法律将少女法律上达到可以自主的年龄从13岁提高到了16岁，包含了一系列措施保护妇女免受利用，诸如在色情场所被下迷药、被拐为娼等犯罪行为的侵害。

这个法律提案最初在下议院多年未获准，后来因为自由主义新闻记者威廉·托马斯·斯特德（1849—1912）关于儿童卖淫的一系列报道文章而变得引人注目。斯特德以从一对父母手中购买他们13岁的女儿为代价，让他这个大胆的揭发报道名噪一时。

在斯特德报道引发的强烈公众舆论下，该提案死灰复燃。第11节是由下议院议员Henry Labouchere在1885年8月6日提出的，而第二天就被加进了该提案中，一星期后，该议案最终通过。当时有一些疑问，质疑第11节是否适合加进这样一部核心是用来保护年轻女子和妇女的法律中。[7]

第11节特别针对男性，而在此之前，法律指的“严重猥亵”行为在英国并不是违法的，至少成年人私下进行是没问题的。甚至在那时，“私下”这样的条款似乎容易造成利用法律进行敲诈勒索。[8]

对于第11节，最著名的受害者是奥斯卡·王尔德（1854—1900），他于1895年被告发。王尔德因此被迫服劳役，这可能加速了他的离世。

到了20世纪50年代，刑罚方法变得多样了。图灵为自己的罪名辩护，法庭最后判处图灵1年缓刑，在此期间图灵必须接受荷尔蒙治疗。

使用荷尔蒙治疗同性恋的实验始于20世纪40年代。起初，同性恋行

为被认为是因为缺少足够的男性荷尔蒙激素，但睾丸激素的实际效果却与预期的相反。接着，雌性荷尔蒙激素被用于男性同性恋，结果似乎更能达到预期的目的。[9]在图灵被定罪的那个年代，这种疗法被称为器官疗法，也称为“化学阉割”，这似乎比任何其他行为更让人觉得羞辱。雌激素治疗让图灵失去了性能力，让他的乳房畸形生长。

在20世纪50年代初被认定为是同性恋可就糟了。在美国，50年代初麦卡锡主义下的“赤色恐惧”很快转变为另一种形式上的政治迫害。在美国国务院工作的共产主义者其实并不多，反倒是政府部门私下里有同性恋倾向的人较多。“20世纪50年代到60年代间，大约有1000名国务院的工作人员因所谓的同性恋行为而被开除”。[10]

理论上，“危险分子”是用来形容有泄漏国家机密倾向的人。但实际上，这个词是“同性恋者”的委婉说法。[11]这种臆断是假定同性恋者容易遭到敲诈而泄漏国家机密。然而，现实中能找到这种假定的最好例子，还得追溯到第一次世界大战之前奥地利情报机构的一位头目，而且这个故事的真实性还一直存疑。[12]

美国政府对同性恋的做法影响到了英国政府。1951年，美国国务院开始建议英国外交部注意政府里面的“同性恋问题”，后来施压英国政府更多地关注可能由同性恋引起的安全问题。[13]

阿兰·图灵的择业自由因此变得很狭窄。政府最高机密的工作，例如战时图灵从事的工作，是绝不可能了，图灵也不可能再一次去美国。1952年的一部美国法律禁止“患有精神错乱人格的外国人”入境，暗指的就是同性恋。[14]

所有的这些足以导致图灵自杀吗？我们不清楚。

在英国的大街小巷以及政府部门，同性恋者的生活变得愈发艰难。当约翰·诺特-鲍尔爵士1953年被任命为伦敦大都市警察局局长时，他发誓要“铲除伦敦所有肮脏的场所”。同年，英国内政部指示要加大对“男性堕落行为”的打击力度。至少，伦敦的当地治安官已经厌倦了对犯罪的纵容态度，打算让罪犯“像过去一样直接被送回监狱里”。1953年年末到1954年年初，报纸的头条都是在宣传某些男性被告发的消息。[15]

是不是图灵与某位男性发生了新的关系，那人反过来威胁敲诈图灵？

或者如他母亲所说，图灵的自杀纯属意外？

关于图灵这一时期的精神状态，一个有说服力的解释竟来自于小说而不是历史文献或者自传资料，这说明我们对这段史实的无知。小说家珍娜·列文（也是巴纳德大学的天文和物理教授）描绘了这个受到难言羞辱的人：

他并不知道该如何表达甚至感受他受到的屈辱。这种屈辱像一个损坏的部件在他的身体里震荡着，喋喋不休，瓦解摧残着他的骨架。这种屈辱感不能得到平息，也不能在内心深处沉淀，那样的话，即使痛苦会化脓，但至少这种屈辱感会被隔离在内心深处，甚至可能治愈。他的心理分析医生即便能帮他减轻这种痛苦，但若没有一段长久的时间持续为他轻抚打磨伤口，这种痛苦的感觉可能会激烈地爆发。这种屈辱就是不能深埋下来。[16]

我们不知道1954年6月7日的晚上发生了什么不一样的故事。我们也不知道是什么驱使图灵在睡觉前，将每晚都要吃的苹果蘸上了氰化物。

第二天早晨，图灵被发现已经过世，年仅41岁。

---

[1] 安德鲁·霍奇斯，*Alan Turing*：*The Enigma*（Simon & Schuster，1983），104。

[2] 马丁·戴维斯，“Why Gödel Didn’t Have Church’s Thesis”，*Information and Control*，Vol. 54，Nos. 1/2，（July/Aug .1982），14。

[3] 库尔特·哥德尔，*Collected Works*，*Volume II: Publications 1938-1974*（Oxford University Press，1990），306。Judson C. Webb的介绍从第292页开始，特别是第297页的对哥德尔所相信的人的思想具有一些与物理大脑分离的存在的确认，对于理解哥德尔的评注很有帮助。另一个分析是Oron Shagrir的“Gödel on Turing on Computability”，http://edelstein.huji.ac.il/staff/shagrir/papers/Goedel_on_Turing_on_Computability.pdf。

[4] 阿兰·图灵，“Computing Machinery and Intelligence”，*Mind*，Vol. LIX，No. 236（October 1950），442。

[5] Arthur C. Clark，*2001: A Space Odyssey*（New American Library，1968），ch. 16。

[6] 文字源于http://www.swarb.co.uk/acts/1885Criminal_Law_AmendmentAct.shtml（20(
年4月访问）。

[7] H. Montgomery Hyde，*The Love That Dared Not Speak Its Name: A Candid History of Homosexuality in Britain*（Little，Brown and Company，1970），134。这本书首次以名为*The Other Love*英国出版。

[8] 同上，136。

[9] 霍奇斯，*Alan Turing*，467-471。

[10] David K. Johnson，*The Lavender Scare: The Cold War Persecution of*

Gays and Lesbians in the Federal Government（University of Chicago Press，2004），76。

[11] Johnson，*Lavender Scare*，7-8。

[12] Johnson，*Lavender Scare*，108-109。

[13] Johnson，*Lavender Scare*，133。

[14] 霍奇斯，*Alan Turing*，474。

[15] Hyde，*The Love That Dared Not Speak Its Name*，214-216。

[16] 珍娜·列文，*A Madman Dreams of Turing Machines*（Alfred A. Knopf，2006），214。

第三部分
判定性问题

第12章 逻辑与可计算性

1958年夏天，中国出生的逻辑学家王浩利用他在牛津大学教学工作的休假时间，来到坐落于纽约州波基普西市的IBM研究实验室，操作当时最先进的IBM 704计算机。他在IBM的打孔卡片上编码了几乎半个世纪前由阿尔弗雷德·诺斯·怀特海和伯特兰·罗素合著的巨著《数学原理》中的定理。例如，定理*11.26在《数学原理》中的符号表示是：

$$^{*}\mathbf{11\cdot 26}.\vdash :.(\exists x):(y).\phi(x,y):\supset:(y):(\exists x).\phi(x,y)$$

王浩在打孔卡片上，将它写成：

11*26/EXAYGXY-AYEXGXY

他还写了三个程序来读取这些卡片的内容，并且通过应用各种等式和将这些陈述转换成公理的推理规则来证明被编码的定理。这些程序的大部分时间都用在读取卡片内容和打印证明步骤的机械过程上。据他估计，证明《数学原理》第*1章到第*5章的220条定理的实际处理时间不超过三分钟。他后来提出的第三个程序改进版本能在大约四分钟内证明第*9章到第*13章的158条定理。[1]

王浩的程序并不是使用计算机进行定理证明的首次尝试。[2]1954年，马丁·戴维斯就使用普林斯顿高等研究院建造的计算机，编写了一个简单的、只包含加法的Presburger算术程序。这是目前已知的第一个由计算机给出的数学证明。戴维斯的这个程序证明了，两个偶数的和仍然是一个偶数。[3]

1957年，艾伦·纽厄尔、约翰·肖和赫伯特·西蒙发表了由他们编写的称为“逻辑理论机器”的程序运行结果，它是为兰德公司制造的JOHNNIAC计算机编写的，这台计算机的命名是为了纪念约翰·冯·诺依曼。[4]他们也使用《数学原理》作为定理的来源。相对于数学逻辑，他们对人工智能更感兴趣，因此编写程序来模拟人证明定理时采用的方法（他们称之为“启发式”的方法）。王浩后来探索的是一种更加算法化的方法，它的效率更高，成功率也更高。

逻辑理论机器的结果是以信件形式通知伯特兰·罗素的，据说他回

复道："得知《数学原理》现在可以采取机械方式来完成，我很高兴。要是我和怀特海早知道能这么做，就不用浪费十年的时间来手工完成了。"[5]

《数学原理》有三卷，约2000页，在数学和逻辑领域都是了不起的成就。在现代文库列举的20世纪一百部最佳非小说类著作中，《数学原理》名列第23。[6]然而，极少有人真的能够下决心阅读这本书。斯蒂芬·克莱尼是阿隆佐·邱奇的学生，后来撰写了《数学概论》（1952）和《数理逻辑》（1967），他坦言自己从来没有读过《数学原理》。[7]在那些自1913年来在数理逻辑领域做出贡献的人中，像他这样的人可能占大多数。

《数学原理》的引言只陈述了一个目标，那就是"完整地列举出数学推理的所有方法和步骤"，这就是我们所熟悉的称为逻辑主义的纲领（也是一种数学哲学），逻辑主义是指把逻辑作为数学其他部分的基础。为了实现这个目标，怀特海和罗素采用了一整套的集合理论和逻辑工具，他们有意限定数学技巧的做法在当时看起来是很荒诞的。

大卫·希尔伯特却不这么认为，他将称为形式主义的数学哲学更紧密地联系在一起。形式主义着重公理化理论，特别是在希尔伯特的纲领中，它更多地强调了诸如一致性、稳定性、完备性和可判定性的概念。

大卫·希尔伯特把《数学原理》的逻辑分解成一个个扩展的子集，并且每个子集都可以单独拿来研究。他这样做一方面是为了教学，另一方面也是为了便于分析。1917年冬天至1918年，他在哥廷根以这种方法为基础教授了一门课程。1928年，大卫·希尔伯特和威廉·阿克曼基于该方法，合著了一本120页的书《数理逻辑原理》，这本书通常称为《希尔伯特和阿克曼》，它也是图灵论文的重点——判定性问题——的来源。

图灵在论文中明确指出，他参考了《数理逻辑原理》和《数学基础》。后者是大卫·希尔伯特与保罗·贝奈斯合著的，该书的第一卷于1934年在柏林出版，以《希尔伯特和贝奈斯》著称。（该书的第二卷在1939年图灵的论文发表之后才出版）。

阅读图灵论文的下一部分需要对希尔伯特和阿克曼发展的数理逻辑理论有一定的了解。在接下来对该逻辑的概述中，我将采用图灵使用的记号，这与《希尔伯特和阿克曼》中所使用的记号很相似。我还将模仿把这个逻辑作为特征的扩展子集的方法，这种方法已经成为了一个标准方法，在阿隆佐·邱奇、斯蒂芬·克莱尼、伊里奥特·门德尔森、赫伯特·恩德滕和许多其他数学家关于数理逻辑的教科书中都可以看到。

首先，我要介绍《希尔伯特和阿克曼》中称作Ausssagenkalkul的概念，它后来翻译成句子演算，不过现在更多地称之为命题演算或命题

逻辑。

然后，我会把这一逻辑推广到《希尔伯特和阿克曼》中所称的engere Funktionenkalkul（受限的函数演算）。在那本书的第二版（出版于1938年）中，它更名为engere Pradilatenkalkul（受限的谓词演算），现在更多地称之为一阶逻辑、一阶谓词逻辑或一阶谓词演算。介绍完一些概念后，我就可以介绍一阶逻辑和二阶逻辑的区别了。《希尔伯特和阿克曼》最初把二阶逻辑叫做erweiterte Funktionenkalkül，后来称之为erweiterte Prädikatenkalkül（扩展的谓词演算）。

命题（句子）逻辑处理所有具有真值的陈述命题，也就是说，我们可以判定这些命题的真假。例如下面的命题：

今天是星期三。

7是一个素数。

天正在下雨。

我妈妈的名字叫做芭芭拉。

10是完全平方数。

其中，一些为真，一些为假，还有一些可能对于我而言是真，对于你而言则是假的。（不要争论！）在命题逻辑中，每个命题都有一致的真值，没有模棱两可。并且，如果我们不那么自称自己很擅长分析除了数学命题之外的任意其他命题，困惑就会越少。

在命题逻辑中，通常使用斜体大写字母表示语句。字母表中前面的字母，如*A*、*B*、*C*等，通常代表有固定真值的特定命题；而后面的字母，如*X*、*Y*、*Z*等，用来表示变量命题。

我们可以使用某些连接词把单个命题组合成更复杂的命题。

第一个这样的连接词用一个小写字母v表示，它出自拉丁语单词vel，意思是“或”。特别地，“或”又可分为“与或”和与之相对的“异或”（拉丁语aut）。当*X*或*Y*中的任意一个为真，或两个均为真时，命题：

$$X \text{ v } Y$$

为真。如下的真值表可以帮助展示*X*和*Y*真值的不同组合情况。

X	***Y***	***X* v *Y***
假	假	假
假	真	真
真	假	真
真	真	真

在不会产生混淆的情况下，允许省略符号v。公式*XY*等价于*X* v *Y*。

注意，我没有使用把两个公式写在同一行并用等号连接的方法来表示这种等价关系。命题演算语言不包含等号，至少在《希尔伯特和阿克曼》制定的命题演算中是不包含的。

我们说一个命题等价于另一个命题，指的是当组成它们的命题有相同的对应真值时这两个命题也有相同的真值。我们使用人类语言或者称为元语言来表达这种等价关系，准确地区分逻辑语言和元语言可以避免混淆。

《希尔伯特和阿克曼》使用元语言缩写aq.（表示德语中的äquivalent）或eq.（表示英语的equivalent）表示等价：

$$X \text{ v } Y \text{ eq. } XY$$

记住，这个缩写不是命题逻辑语言的一部分，严格意义上只是为了方便才使用它。

“与”的概念用符号&表示。公式

$$X \ \& \ Y$$

为真当且仅当*X*和*Y*同时为真，其真值表如下。

X	***Y***	***X* & *Y***
假	假	假
假	真	假
真	假	假
真	真	真

“与”运算常称为合取（conjuction），这一称呼源于and一词的语法作用。据此，“或”运算常称为析取（disjuncton），这是人们不太熟悉的词。

很明显，从真值表可以得到如下的结论：

$$X \text{ v } X \text{ eq. } X$$

$$X \ \& \ X \text{ eq. } X$$

你还可以在一个复合命题中同时使用这两个连接符。在这种情况下，先计算v的值，再计算&的值。你可以使用括号来改变运算顺序，也可以只是为了清晰起见而使用括号。

$$X \ \& \ Y \text{ v } Z \text{ eq. } X \ \& \ (Y \text{ v } Z)$$

这两个命题都不等价于：

$$(X \ \& \ Y) \text{ v } Z$$

例如，当*X*为假，*Y*为真，*Z*为真时，前两个命题都为假，而最后的命题

却为真。

我就不喋喋不休地介绍那些确保括号总是正确配对，以及连接词应出现在正确位置的各种各样的规则了。这些规则有助于合式公式（well-formed formula，或称为wff，发音是woofs）这一概念的形成。true、false、T和F都不是命题逻辑词汇表中的单词，但是为了方便，你可以使用它们来代替命题字母。可以用T表示一个恒为真的命题，使用F表示一个恒为假的命题。从现在开始，我也会在真值表里使用T和F。

从真值表很容易得到下面的等价关系：

$$X \text{ v T eq. T}$$
$$X \text{ v F eq. } X$$
$$X \text{ \& T eq. } X$$
$$X \text{ \& F eq. F}$$

很明显，从真值表中也可以看出下面两种运算满足交换律：

$$X \text{ v } Y \text{ eq. } Y \text{ v } X$$
$$X \text{ \& } Y \text{ eq. } Y \text{ \& } X$$

下面两种运算满足结合律：

$$X \text{ v } (Y \text{ v } Z) \text{ eq. } (X \text{ v } Y) \text{ v } Z$$
$$X \text{ \& } (Y \text{ \& } Z) \text{ eq. } (X \text{ \& } Y) \text{ \& } Z$$

下面两种运算满足分配律：

$$X \text{ v } (Y \text{ \& } Z) \text{ eq. } (X \text{ v } Y) \text{ \& } (X \text{ v } Z)$$
$$X \text{ \& } (Y \text{ v } Z) \text{ eq. } (X \text{ \& } Y) \text{ v } (X \text{ \& } Z)$$

如果把真值表中的F替换为0，把T替换成1，可以看出，合取运算其实等价于两个一位二进制数的乘积；相应地，析取与加法相似。因此，有时把合取称为“逻辑乘”，把析取称为“逻辑加”。然而，使用这些术语会导致不一致，因此不建议使用它们。

合取和析取都是二元运算。唯一的一元运算称为“非”或“否”，使用类似减号的一划来表示它：

X	**-*X***
F	T
T	F

《希尔伯特和阿克曼》通过在字母或表达式的上方加一个横线表示“非”运算，图灵的标注与他们的不同。否定总是被最先计算，并且否定的符号只作用于紧随其后的符号。两个否定会抵销：

$$--X \text{ eq. } X$$

下面是两个非常基本的等价关系：

$$X \vee -X \text{ eq. } T$$

$$X \& -X \text{ eq. } F$$

两个最基本的、同样也是最有趣的逻辑关系把合取、析取与否定结合了起来。它们以19世纪的数学家奥古斯都·德·摩根（1806—1871）的名字命名，叫做德·摩根定律，尽管这个基本概念在很早之前已经为亚里士多德所了解。

$$-(X \vee Y) \text{ eq. } -X \& -Y$$

$$-(X \& Y) \text{ eq. } -X \vee -Y$$

用通俗的语言来表达这些等式，其含义是显而易见的。例如，“现在没有下雨或下雪”可以表示成可($X \vee Y$)，它的意思是“现在没有下雨，也没有下雪”，后者可以表示成$-X \& -Y$。如果有人告诉我：“你当然不是既富又帅的人。”也就是$-(X \& Y)$，我可以得出如下的结论：“我想，我要么贫穷或者丑陋，要么既贫穷又丑陋吧……”这也就是$-X \vee -Y$。

注意，可以把德·摩根定律里的否定符号都写在一边，于是它们可以转换成：

$$X \vee Y \text{ eq. } -(-X \& -Y)$$

$$X \& Y \text{ eq. } -(-X \vee -Y)$$

在“与”和“或”运算的真值表中，将“真”都变成“假”，“假”都变成“真”，可以得到相反运算的真值表。我们把它称为“对偶原理”，这同样也适用于复杂的命题。例如，

$$X \& -Y \vee Z$$

对所有的符号都变成其相应的否定形式，然后交换&和v（原有隐含需要加括号的地方要记得加括号），得到的新命题就是原来命题的否定形式：

$$-X \vee (Y \& -Z)$$

如果你想证明这两个命题确实是彼此否定的，那么可以构造下面的真值表来测试所有的值。

X	***Y***	***Z***	***X* &-*Y* v *Z***	**-*X* v (*Y* &-*Z*)**
F	F	F	F	T
F	F	T	F	T
F	T	F	F	T
F	T	T	F	T
T	F	F	T	F
T	F	T	T	F
T	T	F	F	T

T T T	T	F

最后两列的值完全相反，因此，

$$X \& -Y \vee Z \text{ eq. } -(-X \vee (Y \& -Z))$$

没有专门的“异或”运算符，但是可以采用下面的公式实现这个运算：

$$(X \vee Y) \& -(X \& Y)$$

从下面的真值表可以看出这个命题的工作原理。

X	***Y***	***X*** v ***Y***	***X*** & ***Y***	(***X*** v ***Y***)& - (***X*** & ***Y***)
F	F	F	F	F
F	T	T	F	T
T	F	T	F	T
T	T	T	T	F

除了*X*和*Y*均为真的情况，“异或”与一般的析取运算类似。

如果把德·摩根定律应用到“异或”的后半个表达式上，可以得到

$$(X \vee Y) \& (-X \vee -Y)$$

你可能更喜欢这种对称的表达形式。

计算机电路使用异或运算计算两个二进制数位的加法，并使用合取运算计算进位。[8]

第三个二元运算有点棘手，它称为蕴涵，你可以把$X \rightarrow Y$读作“*X*蕴涵*Y*”或“若*X*为真，则*Y*为真”。预先警告一下，很多人在看到蕴涵运算的真值表时，会产生“这里有点儿错”的反应。

X	***Y***	***X*** → ***Y***
F	F	T
F	T	T
T	F	F
T	T	T

前两条看上去很奇怪。如果*X*为假，为什么无论*Y*的值是真是假，$X \rightarrow Y$的值都为真呢？一种理解的方法是，首先假设$X \rightarrow Y$为真。如果$X \rightarrow Y$为真，并且*X*为真，那么*Y*一定为真。然而，如果*X*的值不为真，那么讨论*Y*的值有什么意义呢？一点意义都没有。*Y*可以是任意值。这就是为什么*X*为假的时候，无论*Y*的值为何，$X \rightarrow Y$都为真的原因。

看看这个命题："如果天下雨，那么水汽凝结。"如果天在下雨，那么这个命题为真；如果天没有下雨，这个命题也为真。该命题只有在天下雨，而水汽又没有凝结的时候才为假。

数理逻辑中经常用到蕴涵式。通常情况下，蕴涵符号的左边是我们已知为真的一个公式。因此，如果我们接下来能够证明命题本身为真，就可以推断蕴涵符号右边的公式为真。

蕴涵不满足交换律和结合律。然而，

$$X \rightarrow Y \text{ eq. } -Y \rightarrow -X$$

我们称第二个命题为反命题。如果天在下雨，那么我会打伞。如果我没有打伞，那么肯定没有下雨。蕴涵和析取之间有一个简单的关系：

$$X \rightarrow Y \text{ eq. } -X \vee \mathrm{Y}$$

换句话说，如果X为假，或者Y为真，那么$X \rightarrow Y$为真。同样，也可以使用合取运算来表示蕴涵：

$$X \rightarrow Y \text{ eq. } -(X \ \& \ -\mathrm{Y})$$

如果蕴涵符号的左边是任意多个命题的合取式，那么这些命题中的任何一个都可以放在蕴涵符号的右边：

$$X \ \& \ Y \rightarrow X$$

$$X \ \& \ Y \rightarrow Y$$

《希尔伯特和阿克曼》描述了一个双条件（即"当且仅当"）运算，并且用一个波浪号表示：

X	***Y***	***X ~ Y***
F	F	T
F	T	F
T	F	F
T	T	T

$X \sim Y$为真，当且仅当X和Y的真值相同。图灵在其论文中没有使用这个双条件运算。然而，它等价于两个方向相反的蕴涵式的合取式，这一结论是有意义的。

$$X \sim Y \text{ eq. } (X\ Y)\&(Y \rightarrow X)$$

如果你认可这样的等价关系是有意义的，那么只有当T→T为真（我们都认可）并且F→F也为真（这个未确定）时它才能起作用。同样，T→F和F→T的真值必须相反。我们都同意T→F必然为假，这也意味着F→T必须为真。这证实了蕴涵运算的真值表的正确性。

假设我给出如下的命题：

$$X \vee Y \vee (-X \& -Y)$$

你能告诉我什么？为真还是为假？你可能会拒绝说，除非你知道X和Y的值，否则判定不了这个命题的真假。接着，我会建议你建立一个真值表来检测所有X和Y的值的组合。这个表如下：

X	***Y***	***X* v *Y* v (-*X* & -*Y*)**
F	F	T
F	T	T
T	F	T
T	T	T

无论X和Y各自什么取值，这个命题总为真。我们把这样的命题叫做重言式，或者说它是永真的。永真命题在数理逻辑中很受欢迎，因为无论单个命题的真值为何，它们的值总为真。

我们来看看另一类命题：

X	***Y***	***X* & *Y* & -*X***
F	F	F
F	T	F
T	F	F
T	T	F

这个命题永不为真，这类命题叫做矛盾式。重言式的否定就是矛盾式，而矛盾式的否定就是重言式。

下面是第三个例子：

X	***Y***	***X* v (*Y* & -*Y*)**
F	F	F
F	T	F
T	F	T
T	T	T

这个命题的值随着X和Y真值的变化，有时为真，有时为假。我们称这类命题为可满足式，指在原子命题真值的某种组合情况下，这类命题的真值可以为真。

永真命题（重言式）也被认为是可满足的。一个命题永真当且仅当这个命题的否定是不可满足的。

对任意一个命题，我们都可以使用真值表来判断它是重言式、矛盾式或者只是可满足式。求真值表的过程是机械的，它不需要任何特别的灵感、洞察力或直觉。如果你是一个计算机程序员，很容易就可以想到编写一段程序，读取命题逻辑中的一个命题，计算它，然后输出词语“重言式”、“矛盾式”或“可满足式”。

基于这一原因，我们说命题演算中的命题是可判定的。存在这样的一个判定步骤，来判定命题逻辑的任一命题的正确性或可满足性。

换句话说，命题演算的判定性问题已经解决了，今天我们都可以早点结束工作回家休息了。

但这并不能说明，真值表总是可行的。假设一个命题中包含100个命题变量，真值表的行数就长达2^{100}，也就是1267650600228229401496703205376，约10^{30}，这是一个非常大的数。即使使用未来处理一行仅需一纳秒（一秒的十亿分之一，也即光传播约一英尺所需的时间）的计算机，处理整个真值表的时间也将长达38万亿年，大约相当于宇宙当前年龄的3000倍。

有一个好消息：如果你把命题限制到55个命题变量，仍然做到一纳秒处理一行，那么整个处理过程只需一年的时间。每增加一个新的变量都会让处理时间加倍。在可计算性和复杂性理论中，处理真值表的计算时间是指数级的，因为它与问题大小成指数相关。

由于这些原因，在判定命题逻辑中命题真假时，解法比真值表更有价值。这些技术通常包括把命题表示成范式，这些范式可以是若干个变元构成的析取项的合取，或者是若干个变元构成的合取项的析取。

这就结束了我对于命题逻辑的匆匆的概述。我们必须继续讨论下去，因为命题逻辑对于许多目的而言还不够充分。它的最大问题就是，我们一直在处理整句的陈述命题，而不能把不同命题的本质联系起来。命题逻辑不能成功地分析亚里士多德的三段论（“所有的人都是凡人；苏格拉底是一个人；因此……”）和刘易斯·卡罗尔设想的简单的连锁推理（“爱吃鱼的猫都是可驯的；没有尾巴的猫都不和大猩猩一起玩；有胡须的猫都爱吃鱼；不可训的猫都是绿眼睛的；有胡须的猫才有尾巴；因此……”[9]）。

通过引入命题函数或谓词（图灵喜欢使用前者，如今后者更流行，我将交替使用这两个术语），可以使逻辑的力量更加强大。谓词这个术语出自于语法，从语法的角度看，可以把一个句子分成主语和谓语。例如，在“警察说出了大实话”这句话中，主语是“警察”，谓语是“说出了大实话”。

引入谓词是将命题逻辑转化为一阶谓词逻辑的第一步。无论我们何时使用谓词，都是想把自己限制在一个特定的域或总体中。在现实生活中，这个域通常是指自然数，把其中的个体作为谓词的参数。

在《希尔伯特和阿克曼》中（图灵论文中也是），谓词看起来很像函数，但它的取值只有真和假。我喜欢用整个单词或多个单词来命名谓词，比如IsPrime。谓词IsPrime的域是由自然数组成的，因此IsPrime(7)的值为真，IsPrime(9)的值为假。可以用小写字母表示这个域中的个体，作为变量使用，如IsPrime(x)。

谓词可以包含多个参数。假设现在的域是由你的朋友组成的，他们每个人都有独一无二的名字。当某x爱上某y，谓词Loves(x, y)为真。例如，Loves(Pat, Terry)与命题“Pat爱上Terry”相同。有些谓词中的参数是可以交换的，不过这个命题的谓词不能交换，因此Loves(Pat, Terry)和Loves(Terry, Pat)不同。

我们可以使用与命题逻辑中相同的连接词来连接谓词。如果他们彼此相爱，则下面的命题：

Loves(Pat, Terry) & Loves(Terry, Pat)

为真。如果他们的感情（无论是什么样的感情）是相互的，则命题

Loves(Pat, Terry) ~ Loves(Terry, Pat)

为真。

下面这个命题又是什么意思呢？

Loves(x, Pat)

表达的意思不是完全清楚。如果这个命题确实有某种含义，我们可以猜测，它表示每个人都爱Pat，也有可能是至少有某个人爱Pat。

为了避免这样的二义性，我们必须同时引入两个量词（quantifier，图灵称之为quantors）。第一个是全称量词，它由置于谓词前、括在括号里的变量组成，形式如下：

(x)Loves(x, Pat)

括号中的x表示“对于所有的x”。如果“对于所有的x，x都爱Pat”，那么上式为真，也就是每个人都爱Pat时，命题为真。如果Pat爱每一个人，那么公式

(x)Loves(Pat, x)

为真。我们经常使用符号符表示全称量词，不过《希尔伯特和阿克曼》以及图灵的论文中都没有采用这种记号。

第二种量词是存在量词，可以翻译为“存在”。《希尔伯特和阿克曼》中使用一个正规的字母E来表示存在量词，不过图灵更喜欢使用较

为常见的表示方式$\exists$。例如，

$$(\exists x)\text{Loves}(x, \text{Terry})$$

表示“存在这样的一个x，x爱Terry”，或者说某个人爱Terry，至少有一个这样的人，哪怕这个人就是Terry。

在一阶逻辑（《希尔伯特和阿克曼》中称作“狭义演算”）中，量词只用来修饰表示域中个体的变量。在二阶逻辑（即“广义演算”）中，量词可以修饰代表命题函数的变量。图灵的论文只涉及一阶逻辑。

如果我们处理的是一个有限总体，那么可以把全称量词表示成一个合取式，而把存在量词表示成一个析取式。例如，假设我们的总体由Pat、Terry和Kim组成，则下式

$$(x)\text{Loves}(\text{Kim}, x)$$

等价于如下的合取式：

$$\text{Loves}(\text{Kim}, \text{Pat}) \,\&\, \text{Loves}(\text{Kim}, \text{Terry}) \,\&\, \text{Loves}(\text{Kim}, \text{Kim})$$

所有的个体谓词必须都为真时，整个命题的值才为真。公式

$$(\exists x)\text{Loves}(\text{Kim}, x)$$

等价于：

$$\text{Loves}(\text{Kim}, \text{Pat}) \vee \text{Loves}(\text{Kim}, \text{Terry}) \vee \text{Loves}(\text{Kim}, \text{Kim})$$

只要其中的一个谓词为真，就可以使得整个命题的值为真。

回想一下德·摩根定律的对偶性，并把它应用到这两个公式中，你就会发现，当引入否定命题时，全称量词和存在量词彼此可以相互转换，你不必对此过分惊讶。例如，如果不是每个人都爱Terry，那么下面两个等价的公式均为真：

$$-(x)\text{Loves}(x, \text{Terry}) \text{ eq. } (\exists x)\text{-Loves}(x, \text{Terry})$$

如果没有人爱Terry，则下面的两个公式均为真：

$$(x)\text{ -Loves}(x, \text{Terry}) \text{ eq. } -(\exists x)\text{ Loves}(x, \text{Terry})$$

类似地，

$$(x)\text{ Loves}(x, \text{Terry}) \text{ eq. } -(\exists x)\text{ -Loves}(x, \text{Terry})$$

右边的公式可以解释成："不存在有人不爱Terry的情况。"与此类似，

$$(\exists x)\ \mathrm{Loves}(x,\ \mathrm{Terry})\ \mathrm{eq.}\ \text{-}\ (x)\ \text{-}\mathrm{Loves}(x,\ \mathrm{Terry})$$

意思是，不存在没有人爱Terry的情况。

美国数学家查尔斯·桑德斯·皮尔斯（1839—1914）在研究逻辑量词理论的时候，使用符号Σ（通常与求和相关）表示存在量词，使用Π（通常与求乘积相关）符号表示全称量词。这些表示方法进一步强调了逻辑和二进制运算之间的联系。

由于这些公式中使用到的*x*都被某个量词修饰，因此称之为约束变量，其作用相当于可变的函数参数。任何不是全称量词或存在量词一部分的变量都是自由变量。在下面的公式中，*x*是约束变量，*y*是自由变量：

$$(\exists x)\mathrm{Loves}(x,\ y)$$

只要不与其他的变量相冲突，自由变量和约束变量都是可以改变的。例如，我们可以把先前公式中的*x*改成*z*：

$$(\exists z)\mathrm{Loves}(z,\ y)$$

这个公式与之前的公式表达完全相同的意思，但是，我们不能把这个约束变量变成*y*，因为*y*会与自由变量的*y*冲突，改变后它将变得完全不同。

同一个公式中不能包含两个命名相同的约束变量和自由变量。我们把一阶逻辑中不包含任何自由变量的公式称作命题。使用"命题"一词来描述包含自由变量的公式是不恰当的。

约束变量是有作用范围的，通常用括号来指示。在下面的命题中，*x*受约束于括号中的整个表达式范围：

$$(x)[\mathrm{Loves}(x,\ \mathrm{Kim})\ \&\ \mathrm{Loves}(x,\ \mathrm{Pat})]$$

注意，这里使用的是方括号而不是圆括号，只是为了使得整个语句可读性更强。这个命题的意思是：每个人都爱Kim和Pat。它和

$$(x)\mathrm{Loves}(x,\ \mathrm{Kim})\ \&\ (x)\ \mathrm{Loves}(x,\ \mathrm{Pat})$$

意思相同。现在，这两个约束变量相互独立，可以改变其中一个的名字：

$$(y)\ \mathrm{Loves}(y,\ \mathrm{Kim})\ \&\ (x)\ \mathrm{Loves}(x,\ \mathrm{Pat})$$

如果某个人既爱Kim又爱Pat，那么命题

$$(\exists x)[\text{Loves}(x, \text{Kim}) \,\&\, \text{Loves}(x, \text{Pat})]$$

的值为真。然而，如果把这两个谓词分开，意思就会改变。

$$(\exists x)\,\text{Loves}(x, \text{Kim}) \,\&\, (\exists x)\,\text{Loves}(x, \text{Pat})$$

现在，如果有某个人爱Kim，并且有某个人爱Pat，则命题为真，这两个人未必是同一个人。

现在把前面几个公式中的合取替换成析取：

$$(x)[\text{Loves}(x, \text{Kim}) \vee \text{Loves}(x, \text{Pat})]$$

如果每个人或者爱Kim，或者爱Pat（或者既爱Kim也爱Pat），则该命题的值为真。如果Terry爱Kim但不爱Pat，或者Terry爱Pat但不爱Kim，命题也均为真。把两个谓词分开以后，命题的意思会发生变化：

$$(x)\,\text{Loves}(x, \text{Kim}) \vee (x)\,\text{Loves}(x, \text{Pat})$$

只有当每个人都爱Kim，或者每个人都爱Pat，或者每个人都爱这两个人的时候，命题才为真。

下面是一个把存在量词应用到析取式中的例子：

$$(\exists x)[\text{Loves}(x, \text{Kim}) \vee \text{Loves}(x, \text{Pat})]$$

它表示存在某个人爱Kim，或爱Pat，或者同时爱这两个人。把这两个谓词分开后，命题的意思保持不变：

$$(\exists x)\text{Loves}(x, \text{Kim}) \vee (\exists x)\,\text{Loves}(x, \text{Pat})$$

下面两个基本的关系适用于所有的命题函数。在这两个例子中，A是谓词，a是定义域中的一个元素。第一个关系是：

$$(x)A(x) \rightarrow A(a)$$

如果这个谓词对定义域中的所有元素都为真，那么它对任意个体的值均为真。第二个关系如下：

$$A(a) \rightarrow (\exists x)A(x)$$

几个量词可以一起使用。例如，

$$(\exists x)(y)\text{Loves}(x, y)$$

它表达的意思与把量词按如下方法组织在一起所表达的意思相同：

$$(\exists x)[(y)\text{Loves}(x, y)]$$

如果存在某个人爱所有的人，则该命题为真。改变量词的顺序，表达的意思就不大一样了：

$$(y)(\exists x)\ \text{Loves}(x, y)$$

如果每个人都被某个人爱，则命题为真，但某个人未必是同一个人。例如，如果Kim爱Kim，Terry爱Terry，但是他们彼此不相爱，则，

$$(y)(\exists x)\ \text{Loves}(x, y)$$

为真，但是，

$$(\exists x)(y)\text{Loves}(x, y)$$

为假。然而，如果以存在量词开头的命题为真，那么另一个以全称量词开头的命题也为真。你在本章开头看到的《数学原理》中的定理*11.26里就包含了这个关系：

$$*11\cdot 26.\vdash :.(\exists x):(y).\phi(x,y) :\supset: (y):(\exists x).\phi(x,y)$$

其中$ø(x, y)$是谓词。图灵采用的记号为：

$$(\exists x)(y)\phi(x,\ y) \rightarrow (y)(\exists x)\phi(x,\ y)$$

当公式中出现一连串全称量词时，可以任意改变它们的顺序，而不会影响公式的含义。对于存在量词也是如此。（可以把命题转换成复合合取或析取形式来证明这个结论。）然而，一般来说，改变一连串交替出现的全称量词和存在量词的顺序会导致公式含义的变化。

正如命题逻辑中一样，在不考虑域和谓词的含义的情况下，可以计算公式。公式

$$(x)[F(x) \text{ v } \text{-}F(x)]$$

是永真的，因为无论域和命题函数F的定义为何，该公式的值永远为真。然而，下面这个公式不可能为真：

$$(\exists x)(F(x)\ \&\ \text{-}F(x))$$

我们称这样的公式是可驳的。还有其他介于两者之间的公式。下面是个简单的例子：

$$(x)F(x)$$

很容易为x想到一个定义域和函数F，使式子为真。例如，假设这个域由自然数组成，F表示“大于或等于零”。定义一个域和函数使得式子为假同样很容易。例如，假设参数是素数时F返回真。这样的公式可以称是可满足的，因为在某些解释下它的值为真。

正确性和可满足性是同一个问题的两个相对面，因为它们的概念是相关的：一个命题或者是可满足的，或者是可驳的。如果一个命题$\mathfrak{A}$正确，则它是可满足的（但是反之不一定）。如果$\mathfrak{A}$是可满足的但并不正确，则-$\mathfrak{A}$同样也是可满足的但不是可驳的。$\mathfrak{A}$是有效的当且仅当-$\mathfrak{A}$不是可满足的。

正确性和可满足性这两个词有时会同数理逻辑的语义方法相结合，之所以这样称呼这些方法，因为它们给出了所涉及命题的真实含义。

数理逻辑的另一种研究方法从本质上说是句法方法。从公理开始推导定理，称这些定理是可证的，因为它们是公理的结果。采用这种句法方法处理逻辑问题，我们就不必牵扯那些麻烦的（可能是形而上学的）“什么是真”的概念。

《希尔伯特和阿克曼》为命题逻辑陈述了四条显而易见的公理，它们出自《数学原理》：

(a) $X \vee X \rightarrow X$

(b) $X \rightarrow X \vee Y$

(c) $X \vee Y \rightarrow Y \vee X$

(d) $(X \rightarrow Y) \rightarrow (Z \vee X \rightarrow Z \vee Y)$

尽管这些公理仅涉及析取和蕴涵，但如果我们把$X \mathbin{\&} Y$定义成 $-(-X \vee -Y)$的缩写形式，则还可以把它们应用到合取中。

对于一阶逻辑，《希尔伯特和阿克曼》中为其增加了两条公理。对于任意谓词F，下面的命题都可以看成是公理：

(e) $(x)F(x) \rightarrow F(y)$

(f) $F(y) \rightarrow (\exists x)\, F(x)$

除了公理，还有下面这些从原始命题得到复杂命题的规则。

1. 替换：在避免约束变量和自由变量相冲突的情况下，可以用一个

公式来替换一个命题变量；如果可以避免冲突，则约束变量和自由变量都可以改变；可以使用公式来代替谓词。

2. 蕴涵：如果公式 $\mathfrak{A}$ 为真，并且公式 $\mathfrak{A} \rightarrow \mathfrak{B}$ 为真，则 $\mathfrak{B}$ 为真。

第二个规则称为演绎推理（modus ponens），它看上去是显而易见的，但是事实上，它必须是公理。你不能推导出它。如果你非得认为你能做到，那么可以看看刘易斯·卡罗尔的短文《龟对阿基里斯说了什么》。[11]

我们可以把任何从这六条公理和两条规则中推导出来的公式都称作定理。推导过程本身称为证明。证明得出的公式称为是可证明的。一条定理即为一个可证明的公式。

例如，如果 $\mathfrak{A}$ 和 $\mathfrak{B}$ 都是定理，则根据公理(c)和规则(1)，我们可以说

$$\mathfrak{A} \vee \mathfrak{B} \rightarrow \mathfrak{B} \vee \mathfrak{A}$$

是可证明的，因此它是一个定理。

这些规则的应用方式有两种：从公理开始，然后使用规则推导定理；或者从一个公式开始，使用规则把它转化成一个公理，这种情况下就可以证明这个公式是一个定理。本章开头讨论的自动证明程序就是从《数学原理》的定理出发，然后通过应用公理和替换规则，最后把它们划归为公理的。

可以看到，希尔伯特将数学形式化，就像是把它归纳为符号操作的机械过程。这对亨利·庞加莱（1854—1912）来说是很显然的。庞加莱写道："想象这样一台机器，我们从一边输入公理，然后从另一边得到定理，就像芝加哥的传奇机器，猪活着进去，出来时就变成火腿和香肠了"。[12]

你甚至可以用系统化的方式机械地列举出所有的定理。从公理出发，可以把它扩展为任意数目的命题变量和谓词，然后在每一种可能的组合情况下应用替换和蕴涵规则。

根据定义，定理就是可以从公理推导出来的公式，因此这种定理的枚举方法可以产生所有可能的定理。这里就出现了一个问题：这些定理和永真的公式相同吗？或者，可能存在不能从公理推导出的永真公式

吗？

以《希尔伯特和阿克曼》一书作为出发点，库尔特·哥德尔于1929年在其博士论文《关于逻辑演算的完备性》中，首次建立了一阶逻辑的语法方法和语义方法之间的等价关系，随后在1930年的《逻辑函数演算中公理的完备性》论文中再次提出该关系。

在哥德尔之前，人们已经知道每个可证明的公式也是永真的。这一性质称为可靠性，对逻辑系统来说是很重要的。哥德尔证明的是，每一个永真的公式也是可证明的。这可以作为逻辑系统“完备性”的一个定义，的确，哥德尔论文的题目提到了“完备性”。哥德尔的完备性定理证明公理系统是完备的——《希尔伯特和阿克曼》一书为单纯的谓词逻辑提出的公理系统，对于枚举出该逻辑中的每一个永真命题来说是足够的。

可以设想，定理的枚举和哥德尔的完备性定理为一阶逻辑的判定过程提供了依据。例如，假设你想判定公式 $\mathfrak{A}$ 是否是可证明的，可以先列举所有的定理，并且把它们与 $\mathfrak{A}$ 相比较。然而，如果 $\mathfrak{A}$ 是不可证明的，你就无法得到匹配，也不知道何时该停下来。

你一定想到了更聪明的做法：列举所有的定理，然后把每个定理及其否定式和 $\mathfrak{A}$ 进行比较（或者把每个定理同时与 $\mathfrak{A}$ 和它的否定式进行比较）。但是，这样做你仍然不能保证可以找到匹配，因为 $\mathfrak{A}$ 可能仅仅是可满足的，而不是永真的或可驳的。因此，基于枚举的判定过程称为半可判定的。只有当你预先知道 $\mathfrak{A}$ 或 $\mathfrak{A}$ 是永真的时候，这类过程才能成功地得到结论。甚至在哥德尔发表了1930年的论文之后，一阶逻辑的判定性问题仍然是个未解决的问题。

1931年，哥德尔发表了更著名的论文，涉及一阶逻辑在基本算术中的应用——加和乘。使用这种算术，哥德尔能够把一个数与每个公式和每个证明都联系起来。他还为“可证明”构造了一个名为Bew[13]的谓词，并把这个谓词应用到其否定式的哥德尔数中，得到断定自身不可证明的公式。

因此，在支持基本算术的逻辑系统中，构造出既不是可证明的又不

是不可证明的命题是可能的。尽管这一理论后来称为哥德尔不完备性定理，但那篇论文的题目其实是《论“数学原理”及其相关系统的形式不可判定命题I》[14]。标题指的不是完备性或不完备性，而是不可判定命题。

哥德尔的不完备性理论对通用判定过程是毁灭性的打击吗？也不一定，尽管与1930年相比，在1931年，这个通用判定过程确实看上去更加不可能存在了。哥德尔的不完备性定理是关于不可判定命题的，而判定性问题关心的是，是否存在通用过程来判定某个给定公式的可证明性。如果存在通用判定过程，那么它可以把一个不可判定的命题归为不可证明的。

证明《数学原理》中定理的早期计算机程序肯定没有采取这种方法，而是从公理出发，系统地推导出所有可证明的公式。纽厄尔—西蒙—肖的论文将这种方法称为“大英博物馆算法”，如此称呼是因为该方法类似于在大英博物馆中检验每一个物品，希望发现你确实想要的那一件。在这种暴力方法刚一提出时，这些早期的研究学者们就反对它，正如马丁·戴维斯所说的：

> 显然，从某个逻辑系统的公理出发，在所有可能的方向上尝试系统地应用推导规则来证明某些“非平凡”事物，肯定会带来巨大的组合爆炸。[15]

只有一位编程者不担心组合爆炸，他就是阿兰·图灵。图灵的虚拟计算机拥有无限的存储空间和充裕的时间，因此他可以尝试其他更多担心受机器限制的程序员们所不敢触及的问题。

在前一章中，我在第9节“可计算数的范畴”的中间部分停了下来。图灵在第9节开始就要我们相信通过他的机器进行计算的数包括了“所有被自然当做可以计算的数字”（图灵论文第249页，本书第176页）。

然后，图灵以罗马数字I（表示几个论点中的第一个）和“类型(a)”（直觉的引导）为标题开始了第一小节。下一小节以罗马数字II和“类型(b)”开头，这是图灵的第二个论点。“类型(b)“是指”两种定义等价的证明（新的定义有更多的直觉成分）”。

标题后的一句话有三个脚注。第一个只是阐明他要论述的是受限的谓词演算，也就是我们所说的一阶谓词逻辑。暂时忽略第二个脚注，我们会尽快讨论它。

> II. [Type (*b*)].
>
> If the notation of the Hilbert functional calculus† is modified so as

to be systematic, and so as to involve only a finite number of symbols, it becomes possible to construct an automatic[‡] machine, $\mathcal{K}$, which will find all the provable formulae of the calculus[§].

[†] The expression “the functional calculus” is used throughout to mean the *restricted* Hilbert functional calculus.

[‡] It is most natural to construct first a choice machine (§ 2) to do this. But it is then easy to construct the required automatic machine. We can suppose that the choices are always choices between two possibilities 0 and 1. Each proof will then be determined by a sequence of choices $i_1, i_2, \ldots, i_n$ (i_1 = 0 or 1, i_2 = 0 or 1, …, i_n = 0 or 1), and hence the number $2^n + i_1 2^{n-1} + i_2 2^{n-2} + \ldots + i_n$ completely determines the proof. The automatic machine carries out successively proof 1, proof 2, proof 3, …

[§] The author has found a description of such a machine.

II. [类型(b)]

如果修改希尔伯特的谓词演算[†]中的记号，使它们系统化并且只涉及有限的符号，那么就有可能构造一台自动[‡]机器 $\mathcal{K}$ ，用来寻找演算过程中所有可证明的公式[§]。

[†] “谓词演算”表示受限的希尔伯特谓词演算。

[‡] 最自然的事情就是首先构造一台选择机器（§2）来完成这个工作，然后就可以很容易构造想要的自动机器了。我们假设总是在两个可能性0和1之间进行选择。这样，每个证明都由一个选择序列$i_1, i_2, \ldots, i_n$（i_1= 0或1, i_2=0或1,…, i_n=0或1）决定，因此，数$2^n + i_1 2^{n-1} + i_2 2^{n-2} + \ldots + i_n$就完全决定了证明。自动机器依次执行证明1、证明2、证明3……

[§] 作者发现了对这样一台机器的描述。

我相信，图灵把机器称作$\mathcal{K}$是用来代表德语单词Kalküi。尽管这一点在这句话中不是很明显，但是最后你会发现图灵描述的是一个大英博物馆算法。机器$\mathcal{K}$可以从已经编码在磁带上的公理开始，也可以一开始就在磁带上写下公理。这些公理包括一阶逻辑中的基本公理以及谓词需要的其他公理。机器逐步地执行推导规则来产生演算中所有可证明的命题。

图灵要求修改一阶逻辑的记号来“使它们系统化”。毫无疑问，我们不想困扰于仅仅是变量名称或不必要的括号使用所造成的不同命题间的等价性。例如，下面的三个命题是完全相同的：

$$(x)(\exists y)\phi(x, y)$$

$$(y)(\exists x)\phi(y, x)$$

$$(x)(\exists y)(\phi(x, y))$$

使记号系统化的一个方法是要求变量总是以x_i的形式出现，并且它们在某个特定公式中出现时必须遵循下标的数字顺序。另外，括号只有在需要调整运算顺序时才能使用。另一种方法（它与实际机器中采用的方法更类似）是把公式编码成前缀形式，这样就没有必要使用括号，比如卢卡西维茨（1878—1956）专门为命题逻辑发明的所谓的波兰表示法。命题

$$(A \text{ v } B) \ \& \ (C \text{ v } D)$$

就可以编码成：

$$\& \text{v} AB \text{v} CD$$

这里重要的是，这台机器要产生所有可证明的公式。从哥德尔的完备性定理中，我们知道这些所有可证明的公式集合和所有永真的公式集合是相同的。

我相信，图灵在试图吸引那些其论文的早期读者，他们可能对图灵的机器可以计算任意复杂度的实数的能力表示怀疑。1936年，人们对一阶逻辑效力的信任超过了计算机器。从实施的角度看，机器$\mathcal{K}$看上去很可行，它确实比计算实数（如10的7次方根）的机器要简单很多。

机器$\mathcal{K}$仅采用字符串工作，并根据替换规则以各种方式将这些符号组织在一起。对于很多字符串比较和替换逻辑，图灵在其通用机中所使用的函数里已经有所体现了。

这一句中的第二个脚注——以“最自然的……”开头的脚注，其实看上去是要描述另一种不同的方法，与一阶逻辑相比，这种方法更适合于命题逻辑。给出固定数目的n个命题变量，你可以构造一个系统来产生所有由这些变量和逻辑连接词交替构成的合式公式。对其中的每个公式，都可以使用真值表方法来判断其是否是一个重言式。

如果这个合式公式包含n个命题变量，则需要2^n次测试来决定它是否是正确的。如果把真和假分别看作二进制数字0和1，则每次测试都与一个n位的二进制数对应，该二进制数中的每一位都表示一个变量的真值。图灵脚注里的二进制数有点错误，应该去掉开头的2^n项。这些试验可以从0开始编码一直到n-1，对应n位二进制数的值。

尽管图灵需要使用这台机器$\mathcal{K}$来产生一阶逻辑而不是命题逻辑中的命题，但是我们可以发现，无论何时需要整数，他都只要求一个由非负整数构成的有限的域。他从来不要求一个无限的域，因此不难想象，我们可以把他的一阶公式转换成命题公式，然后就能使用真值表的方法了。

把自然数引入一阶逻辑系统总是有点杂乱，但是如果要把这个逻辑应用到数值概念中，那就非常有必要了。把数字合并进逻辑通常从皮亚诺公理出发。这五个公理来自1889年，朱塞佩·皮亚诺在他的一本小册子《算术原理：一种新的方法》[15]中提出了最初的九条公理，皮亚诺公理就是从这些公理中提取出来的，它的提出也基于了理查德·戴德金1888年出版的一本小册子《什么是数字，什么应该是数字》[16]。

皮亚诺公理是围绕“后继”概念建立的。“后继”就是直觉上一个数后面紧跟的下一个数。例如，12的后继是13。皮亚诺公理保证每一个数都有一个后继，并且该后继是唯一的。在皮亚诺的构造中，只有一个数不是其他任何一个数的后继，这个数是1。不过现在，一般情况下我们定义自然数是从0开始的。

下面是皮亚诺公理的一种通俗易懂的版本：

1. 零是一个数；
2. 每个数都有一个后继，它也是数；
3. 零不是任意数的后继；
4. 后继数相同的两个数是相等的；

5. 任意关于自然数的命题，如果它对零为真，并且它对某个数为真就意味着它对该数的后继也为真，则该命题对所有的数均为真。

第5条公理通常称为数学归纳原理，它是很多涉及自然数的数学证明的基础。（在后两章中，图灵会在证明中两次使用到它。）然而，用一阶逻辑语言表达归纳法是有问题的。归纳法是二阶逻辑固有的概念，因为它必须适用于所有包含自然数参数的谓词。相等也是二阶逻辑的概念，这正是逻辑学家（及图灵在其论文中）不愿意引入在两个参数相等时值为真的谓词的原因。

即使在一阶逻辑语言中表达前四条皮亚诺公理也不是一件容易的事，而且（正如你将看到的）图灵的表示也是不充分的。这个问题对他的证明或结论没有什么实际影响，不过它确实令人不安。

另一个问题涉及自然数本身的表示。传统上使用0, 1, 2, 3,…的奇特表示方法不会产生这个问题。几个世纪的惯例以及小学课程的强行灌输告诉我们，12的后继就是13。特别地，如果我们构思符号操作的机械形式，更好的做法是让符号本身传达“后继关系”的概念。

在《数学基础》（1934）的第一卷中（图灵在论文中引用的一本书），保罗·贝奈斯使用“′”来表示一个数是另一个数的后继。例如，如果a是一个数，则a'表示该数的后继，a''表示下一个后继。贝奈斯还使用符号0来表示零，此时，$0'$是0的后继，$0''$是再下一个后继。[17]它究竟是哪个数字呢？只需要数一下0上“′”的个数就可以知道。在希尔伯特及其追随者的著作中，我最早看到这种标注方式的是大卫·希尔伯特1927年发表的《数学基础》。[18]

图灵没有完全使用这种方式。显然，他甚至不太愿意使用符号0，而是使用符号u表示第一个自然数。（他并没有交代u究竟是0还是1，即使这很重要。在我的例子中，我都假设u表示0。）u的后继则为u'、u''、u'''等。在表示大数时，这种记号就不太灵活了，并且它不能表示如n或x的任意数。受到《希尔伯特和贝奈斯》的启发，图灵使用$u^{(r)}$表示u的右上方有r个“′”。例如，u'''''可以表示成$u^{(5)}$，我们知道这就是一只手的手指个数。

图灵定义了一个命题函数$N(x)$，当x为非负整数时，该函数的值为真。如果我们总是把自己限制在非负整数的范围内，那么这个函数对我们没有任何意义，然而，图灵发现可以使用它表示皮亚诺公理。

图灵还定义了一个命题函数$F(x, y)$，当y是x的后继，或者用普通算术描述为$y = x + 1$时，该函数的值为真。记住，F并没有提供或计算出这个后继。它是程序员们所谓的布尔函数，只有在y确实是x后继的情况下，才为真。

一旦定义了一个好的后继谓词（例如命名为Succ，这样命名仅仅是

为了和图灵定义的谓词区分开来），并且为数学归纳法的使用建立了一个公理，那么就可以定义一个名为Sum(x, y, z)的谓词，当z的值与$x + y$的值相等时，这个谓词的值为真。这个谓词Sum基于以下三条公理：

$$(x)\mathrm{Sum}(x, u, x)$$

$$(x)\mathrm{Sum}(u, x, x)$$

$$(x)(y)(z)(r)(s)(\mathrm{Sum}(x, y, z)\&\mathrm{Succ}(y, r)\&\mathrm{Succ}(z, s)\rightarrow\mathrm{Sum}(x, r, s))$$

前两条公理定义了零和任意一个数的加法运算。第三条的含义是：如果$x + y = z$，$r = y + 1$，$s = z + 1$，那么$x + r = s$。

类似地，可以定义谓词Product(x, y, z)如下：

$$(x)\ \mathrm{Product}\ (x, u, u)$$

$$(x)\ \mathrm{Product}\ (u, x, u)$$

$$(x)(y)(z)(r)(s)(\ \mathrm{Product}\ (x, y, z)\ \&\ \mathrm{Succ}(y, r)\ \&\ \mathrm{Sum}(z, x, s)\ \rightarrow\ \mathrm{Product}\ (x, r, s))$$

前两条公理定义与零的乘法运算，第三条的含义是：如果$x \times y =$ z，$r = y + 1$，$z + x = s$，那么$x \times r = s$。

我们再做进一步的分析。定义如下的谓词IsEven(x)：

$$(\exists y)\ \mathrm{Product}(y, u'', x)$$

如果存在这样的y满足$x = y \times 2$，那么IsEven(x)为真。IsOdd(x)与-IsEven(x)相同。我们已经有了足够多的谓词，因此这里不再继续定义了。

Now let α be a sequence, and let us denote by $G_\alpha(x)$ the proposition "The x-th figure of α is 1", so that — $G_\alpha(x)$ means "The x-th figure of α is 0".

The negation sign is written before an expression and not over it.

令α是一个序列，我们用$G_\alpha(x)$表示命题"α的第x个数字是1"，于

是，$-G_\alpha(x)$ 表示“α的第x个数字是0”。

否定符号写在表达式的前面，而不是上面。

上面的脚注表明，图灵采用的否定记号和希尔伯特的不同。α是一个序列，这个序列可以通过设计一个专用机器按常规计算得到。这里，图灵建议我们根据数字1或0来对应建立值为真或假的谓词。例如，第6节中得到的对应于2的平方根的序列以

1011011010…

开头。

如果从0开始对每一位上的数进行编号，那么$G_\alpha(0)$的值为真，$G_\alpha(1)$的值为假，$G_\alpha(2)$的值为真，$G_\alpha(3)$的值为真，$G_\alpha(4)$的值为假，等等。这个谓词好像很复杂。图灵例I中机器对应的序列要简单得多：

0101010101…

你可以回想起这个序列正是1/3的二进制等价形式。如果将$G_\alpha(x)$定义为IsOdd(x)，则描述这个序列的命题函数是很简单的。

> Suppose further that we can find a set of properties which define the sequence α and which can be expressed in terms of $G_\alpha(x)$ and of the propositional functions $N(x)$ meaning “x is a non-negative integer” and $F(x, y)$ meaning “$y = x + 1$”.

> 进一步假设我们可以找到一组性质来定义序列α，并且可以用$G_\alpha(x)$和命题函数$N(x)$、$F(x, y)$来表达这些性质，其中，$N(x)$表示“x是一个非负整数”，$F(x, y)$表示$y = x + 1$。

图灵在这里引入的两个谓词将在论文的剩余部分中经常用到。F是一个后继函数，它对于自然数的定义很重要。如果定义域明确限制为自然数，我认为不需要定义函数N。

我不确定图灵所说的“一组性质”指的是什么。我们这里需要的是能为构成$G_\alpha(x)$的命题函数提供支持的公理。在我给出的简单例子中，这些

公理包括支持谓词Sum和Product的公理。

When we join all these formulae together conjunctively, we shall have a formula, $\mathfrak{A}$ say, which defines α.

用合取符号把这些公式连接起来，我们可以得到一个定义α的公式，将其命名为$\mathfrak{A}$。

公理的合取式不一定真正地定义了α，因为它无法为定义α提供基础。

The terms of $\mathfrak{A}$ must include the necessary parts of the Peano axioms, viz.,

$$(\exists u)\ N(u)\ \&\ (x)\ (N(x) \rightarrow (\exists y)\ F(x, y))\ \&\ (F(x, y) \rightarrow N(y)),$$

which we will abbreviate to P.

$\mathfrak{A}$的项中必须包含皮亚诺公理的必要条件，也就是：

$$(\exists u)N(u)\&(x)(N(x)\rightarrow(\exists y)F(x,y))\&(F(x,y)\rightarrow N(y))$$

后文中我们将其简写作P。

当然，P指的是皮亚诺，它是三个项的合取式。第一项表示u存在，第二项的含义是对每一个x都存在一个y是它的后继，第三项指的是一个

数的后继也是一个自然数。这个公式不能说明零和后继的唯一性，这是个问题。《希尔伯特和贝奈斯》中为后继函数（他们称之为S[19]）给出了以下三个公理：

$$(x)(\exists y)S(x, y)$$

$$(\exists x)(y) - S(y, x)$$

$$(x)(y)(r)(s)(S(x, r) \& S(y, r) \& S(s, x) \rightarrow S(s, y)$$

第一条公理声明每个数都有一个后继；第二条的含义是存在一个没有后继的数；第三条是说，如果r是x和y的后继，x是s的后继，那么y也是s的后继。第三条公理实质上确立了后继的唯一性。

When we say "$\mathfrak{A}$ defines α", we mean that — $\mathfrak{A}$ is not a provable formula, and also that, for each n, one of the following formulae (A_n) or (B_n) is provable.

$$\mathfrak{A} \ \& \ F^{(n)} \rightarrow G_\alpha(u^{(n)}), \qquad (A_n)¶$$

$$\mathfrak{A} \ \& \ F^{(n)} \rightarrow (-G_\alpha(u^{(n)})), \qquad (B_n),$$

where $F^{(n)}$ stands for $F(u, u') \& F(u', u'') \& \ldots F(u^{(n-1)}, u^{(n)})$.

¶ A sequence of r primes is denoted by $^{(r)}$.

当我们说"$\mathfrak{A}$定义了α"时，意思是-$\mathfrak{A}$不是一个可证明的公式，并且对每一个n，下面公式(A_n)或(B_n)中的一个是可证明的。

$$\mathfrak{A} \ \& \ F^{(n)} \rightarrow G_\alpha(u^{(n)}), \qquad (A_n)¶$$

$$\mathfrak{A} \,\&\, F^{(n)} \to \left(-G_{\alpha}(u^{(n)})\right), \qquad (\mathrm{B}_n)$$

其中$F^{(n)}$代表$F(u,u')\&F(u',u'')\&\ldots F(u^{(n-1)},u^{(n)})$。

¶ 包含r个“′ ”的序列记为$^{(r)}$。

公式(A_n)上的脚注是指，按照惯例采用括号中含数作为上标，来表示包含相应数目的“′”。图灵在其后继函数F中也使用上标来表示后继函数的合取，后继函数实际上是说，1是0的后继，2是1的后继，等等。

图灵采用的后继函数的合取是不充分的，因为它不能保证这些后继的唯一性。例如，$F(u', u''')$的真值是多少？我们无法得知它是假的。一种简单的修正方式是通过包含所有不为真的后继谓词的否定式，如$-F(u', u'')$和$-F(u', u)$，并在$u^{(n)}$处停止来扩展$F^{(n)}$（尽管仍是有限的）。

这里的n指的是位数，它从0开始逐个变大。从0位，1位，2位，……逐位生成序列。每一位的计算都只需要有限数目的非负整数，因此公式$F^{(n)}$是有限个项的合取。然而，在某些情况下，公式可能需要更多的整数。例如，对于第0位，公式只需要u。但是在我的例子中，函数IsOdd的定义还需要u''，因此事实上，F的上标应该是2和n中较大的那个数。

进行这个小小的修正后，下面的公式都将是可证明的：

$$B_0 : \mathfrak{A} \,\&\, F^{(2)} \to -\mathrm{IsOdd}(u)$$

$$A_1 : \mathfrak{A} \,\&\, F^{(2)} \to \mathrm{IsOdd}(u')$$

$$B_2 : \mathfrak{A} \,\&\, F^{(2)} \to -\mathrm{IsOdd}(u'')$$

$$A_3 : \mathfrak{A} \,\&\, F^{(3)} \to \mathrm{IsOdd}(u''')$$

$$B_4 : \mathfrak{A} \,\&\, F^{(4)} \to -\mathrm{IsOdd}(u'''')$$

$$A_5 : \mathfrak{A} \,\&\, F^{(5)} \to \mathrm{IsOdd}(u''''')$$

依次类推。回想一下，$\mathfrak{A}$包括支持IsOdd函数所需要的所有公理。这些结果对应了序列的前六位：0，1，0，1，0，1。注意，数字1对应可证明的A_n，0对应B_n的可证明性。

[253]

I say that α is then a computable sequence: a machine $\mathcal{K}_\alpha$ to compute α can be obtained by a fairly simple modification of $\mathcal{K}$.

我认为此时的α是一个可计算序列：稍微修改$\mathcal{K}$，便得到一个可以计算α的机器$\mathcal{K}_\alpha$。

大家会想起，机器$\mathcal{K}$可以从公理推导产生所有的可证明的公式。

We divide the motion of $\mathcal{K}_\alpha$ into sections. The n-th section is devoted to finding the n-th figure of α. After the (n - 1)-th section is finished a double colon :: is printed after all the symbols, and the succeeding work is done wholly on the squares to the right of this double colon. The first step is to write the letter “*A*” followed by the formula (A_n) and then “*B*” followed by (B_n).

我们把$\mathcal{K}_\alpha$的操作分成几个部分。第n个部分用来寻找序列α的第n位数字。第(n-1)个部分完成以后，在所有符号的末尾打印一个双冒号::，后续的工作全都在这个双冒号右边的格中进行。第一步是写入字母“A”，后面跟着写公式A_n；然后写入“B”，再紧跟着写

B_n。

在这个例子中，n的值为5，机器首先写“A”和“B”，紧跟着分别写入两种可能的情况，如下：

$$\mathbf{A}\,\mathfrak{A}\ \&\ F^{(5)} \rightarrow \mathrm{IsOdd}(u'''''')\ \mathbf{B}\,\mathfrak{A}\ \&\ F^{(5)} \rightarrow -\mathrm{IsOdd}(u'''''')$$

不过，上面的写法也不完全正确：字母“A”和“B”不应该为粗体，项 $\mathfrak{A}$ 应该是所有公理的显式合取式，$F^{(5)}$是更多个公理的显式合取式，IsOdd可能是之前定义的Product函数的否定式，所有函数的名字可能变得更加神秘了。

然而，真正的意义在于，这两个命题中的其中一个是可证明的。在这两个打印的公式右边的全部有一整条纸带可以供机器工作。很可能是机器首先把这些公理写在纸带上，然后开始推导可证明的公式。

The machine $\mathcal{K}_\alpha$ then starts to do the work of $\mathcal{K}$, but whenever a provable formula is found, this formula is compared with (A_n) and with (B_n). If it is the same formula as (A_n), then the figure “1” is printed, and the n-th section is finished. If it is (B_n), then “0” is printed and the section is finished. If it is different from both, then the work of $\mathcal{K}$ is continued from the point at which it had been abandoned. Sooner or later one of the formulae (A_n) or (B_n) is reached; this follows from our hypotheses about α and $\mathfrak{A}$, and the known nature of $\mathcal{K}$. Hence the n-th section will eventually be finished. $\mathcal{K}_\alpha$ is circle-free; α is computable.

随后，机器$\mathfrak{K}_\alpha$开始从事$\mathfrak{K}$的工作，但是无论何时找到一个可证明的公式，这个公式都会与A_n和B_n进行比较。如果它和A_n相同，就打印数字1，这样第n个部分就完成了。如果它和B_n相同，则打印0，这个部分也就结束了。如果它和A_n、B_n都不相同，那么$\mathfrak{K}$就从它停止的点继续工作，迟早会遇到A_n或B_n中的一个。这可以从我们对α和$\mathfrak{A}$所作的假设，以及$\mathfrak{K}$的性质中推断得到。因此，第n个部分最终会结束。$\mathfrak{K}_\alpha$是非循环的，α是可计算的。

可以想象，机器$\mathfrak{K}_\alpha$可以一般化为与图灵的通用机极其类似的机器。它可以以一条已经编码了公理的纸带开始工作，仅仅通过在纸带上写入一些不同的公理和一个不同的函数，机器就能计算通过一阶逻辑定义的任意序列。

It can also be shown that the numbers α definable in this way by the use of axioms include all the computable numbers. This is done by describing computing machines in terms of the function calculus.

还可以证明，通过这种使用公理的方式得到的可定义的数字α包含了所有的可计算数，借助函数演算描述计算机器就可以做到。

事实上，在图灵论文的最后一节，即本书的下一章中，图灵将采用一阶逻辑描述计算机器。现在，他想提醒读者，不是每一个数字都可以通过机器计算得到，特别是那些以0和1构成的序列，它们可以告诉我们哪些是符合要求的机器的描述数。

It must be remembered that we have attached rather a special meaning to the phrase “ $\mathfrak{A}$ defines α”. The computable numbers do not include all (in the ordinary sense) definable numbers. Let δ be a sequence whose n-th figure is 1 or 0 according as n is or is not satisfactory. It is an immediate consequence of the theorem of § 8 that δ is not computable. It is (so far as we know at present) possible that any assigned number of figures of δ can be calculated, but not by a uniform process. When sufficiently many figures of δ have been calculated, an essentially new method is necessary in order to obtain more figures.

必须牢记，我们为“ $\mathfrak{A}$ 定义α”这句话赋予了一个特别的含义。可计算数不包括所有（一般意义上）的可定义数。假设δ是一个序列，根据n是否可接受，它的第n位数字可以为1或0。由§8中的定理可以直接得到δ是不可计算的。（就我们现在所知）给定数目的δ中的数字是可计算的，但是无法采用统一的过程。一旦已经计算了足够多的δ的数字后，有必要采用一个本质上更新颖的方法来得到更多的数字。

至此，图灵完成了他的第二个论证，来证明他的机器可以计算通常所谓的可计算数。接下来将要讲述的是他的第三个论证。你或许能回忆起图灵对于一台人类计算者的“思维状态”的依赖。一些读者可能认为，人类“思维状态”是太过虚幻的概念而不能封装在一台机器中。

图灵简单地描述了如何将思维状态这一概念构造成为机器结构的一部分，我们将以此结束本章。

III. This may be regarded as a modification of I or as a corollary of II.

We suppose, as in I, that the computation is carried out on a tape; but we avoid introducing the “state of mind” by considering a more physical and definite counterpart of it. It is always possible for the computer to break off from his work, to go away and forget all about it,

and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of the “state of mind”. We will suppose that the computer works in such a desultory manner that he never does more than one step at a sitting. The note of instructions must enable him to carry out one step and write the next note. Thus the state of progress of the computation at any stage is completely determined by the note of

[254]

instructions and the symbols on the tape. That is, the state of the system may be described by a single expression (sequence of symbols), consisting of the symbols on the tape followed by Δ (which we suppose not to appear elsewhere) and then by the note of instructions. This expression may be called the “state formula”. We know that the state formula at any given stage is determined by the state formula before the last step was made, and we assume that the relation of these two formulae is expressible in the functional calculus. In other words, we assume that there is an axiom $\mathfrak{A}$ which expresses the rules governing the behaviour of the computer, in terms of the relation of the state formula at any stage to the state formula at the preceding stage. If this is so, we can construct a machine to write down the successive state formulae, and hence to compute the required number.

III. 这一部分可以看成是对I的修正或II的推论。

与I中类似，我们假设计算在一个纸带上进行，但是通过考虑更加机械而确定的类似过程，我们可以避免引入“思维状态”这一概念。计算机总是能在某个时刻暂停，转向其他的操作，然后过一段时间后还可以再返回到之前停止的点继续运行。如果机器这样运行，那么它必须记录一些指令（以某种标准形式写入纸带），以便阐明如何继续当前已经停止的工作。这个记录即与“思维状态”类似。我们假设计算机将以这种不连贯的方式运行，它每次不能执行超过一个步骤。记录的指令必须使得计算机可以执行一个步骤，并且写入下一条记录。因此，计算过程中任意阶段的状态都是由记录的指令和纸带上的符号完全决定的。也就是说，可以采用单一的表

达式（符号序列）来描述系统的状态，这个表达式是由符号、之后的Δ（假设Δ不会在其他地方出现）和指令记录组成的，可以称这个表达式为“状态公式”。我们知道，每个给定阶段的状态公式都是由上一个步骤进行之前的状态公式决定的，我们假定这两个公式之间的关系可以采用函项演算来表示。换句话说，假定存在一个公理$\mathfrak{A}$，它给出了任意阶段的状态公式和前一阶段的状态公式之间的关系，从而管理计算机的行为。如果真是这样，我们可以构造一台机器记下连续的状态公式来计算得到所需的数。

---

[1] 王浩，“Toward Mechanical Mathematics”，*IBM Journal of Research and Development*，Vol. 4，No. 1（Jan. 1960），2-22。可在http://www.research.ibm.com/journal/rd/04l/ibmrd0401B.pdf获得。

[2] Donald Mackenzie，“The Automation of Proof：A Historical and Sociological Exploration”，*Annals of the History of Computing*，Vol. 17，No. 3（1995年秋），7-29。

[3] 马丁·戴维斯，“A Computer Program for Presburger’s Algorithm” in Jörg Seikmann and Graham Wrightson，eds., *Automation of Reasoning 1: Classical Papers on Computational Logic, 1957-1966*（Springer-Verlag，1983），41-48。

[4] Allen Newell，J. C. Shaw，and H. A. Simon，“Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics”，*Proceedings of the Western Joint Computer Conference*，Vol. 15（1957），218-239。重印于Edward A. Feigenbaum and Julian Feldman，eds.，*Computers and Thought*（MIT Press，1995），109-133。

[5] 引用自Michael J. Beeson，“The Mechanization of Mathematics”，in Christof Teuscher，ed.，*Alan Turing: Life and Legacy of a Great Thinker*（Springer，2004），93。

[6] http://www.randomhouse.com/modernlibrary/100bestnonfiction.html。

[7] William Aspray，The Princeton Mathematics Community in the 1930s: An Oral-History Project。1984年4月26日于威斯康星麦迪逊对J. Barkley Rosser和Stephen C. Kleene的采访。http://www.princeton.edu/~mudd/finding_aids/mathoral/pmc23.htm。

[8] 本书作者，*Code: The Hidden Language of Computer Hardware and Software*（Microsoft Press，1999），第12章。

[9] 刘易斯·卡罗尔，*Symbolic Logic: Part I. Elementary*（Macmillan，1896），118。答案是：“没有一只绿眼睛的猫会和大猩猩玩。”你知道的。

[10] 刘易斯·卡罗尔，“What the Tortoise Said to Achilles”，*Mind*，New Series，Vol. 4，No. 14（Apr 1895），278-280，之后被多次重印。

[11] 亨利·庞加莱，*Science and Method*，英译本译者为Francis Maitland（Thomas Nelson & Sons，1914；Dover，2003），147。

[12] Bew是德语单词beweisbar的缩写，意为“可证明的”。——编者注

[13] 我引用的三篇哥德尔的文章可在Kurt Gödel，*Collected Works: Volume I, Publications 1929-1936*（Oxford University Press，1986）得到。

[14] 马丁·戴维斯，“The Early History of Automated Deduction”. in Alan Robinson and Andrei Voronkov, eds.，*Handbook of Automated Reasoning*（MIT Press，2001），Vol. 1，3-15。

[15] 引自 Jean van Heijenoort，ed.，*From Frege to Gödel*：*A Source Book in Mathematical Logic, 1879-1931*（Harvard University Press，1967），83-97。完整版本可在*Selected Works of Giuseppe Peano*，由Hubert C. Kennedy翻译并编辑（George Allen & Unwin，1973），101-134得到。

[16] 重印于William Ewald，ed.，*From Kant to Hilbert*：*A Source Book in the Foundations of Mathematics*（Oxford University Press，1996），Vol. II，787-833。

[17] 希尔伯特和贝奈斯，*Grundlagen der Mathematik*，Volume I（Springer，1934），218。

[18] 希尔伯特，“Foundations of Mathematics”，*From Frege to Gödel*，467。

[19] 希尔伯特和贝奈斯，*Grundlagen der Mathematik*，Volume I，209。我把记号修改了一下，以便和图灵的一致。

第13章 可计算函数

你上一次使用个人计算机计算无理数的无穷数位是在什么时候？除非你是通过运行程序计算你的数百万位数字来消遣，否则你采用的任何程序计算出的数位都不可能超出你喜爱的计算器的计算范围。

很显然，图灵在他的论文中建立了很多计算机编程的原理和概念，但是无论是在以前、现在或是将来，计算实数的无穷数位都必然不是计算机的典型活动。

实际上，计算机执行的是已经被程序员分割成若干组块的任务，这些块称作函数、例程、子程序或方法（称呼取决于特定的编程语言）。一般来说，这些函数在有限的时间段内执行某个特定的任务，它们从接收输入开始，然后对这些输入进行数学运算以产生输出，最后把控制转移给其他函数。

函数的概念起源于数学。概括地说，函数是把输入转换成输出的数学实体。输入称作函数的参数或自变量；输出称作函数的值或因变量。函数通常被限制在特定类型的数字或其他对象上。合法的输入称作函数的定义域，所有可能的结果输出值称作值域。

图灵在其论文的第一段提到了“可计算函数”的概念，并把它作为未来探索的主题：

> 尽管本论文的主题是可计算数，但是我们同样可以容易地定义和研究可计算函数，其变量可以是整数、实数、可计算数、可计算谓词等。……我希望可以很快就给出可计算数、函数等之间的关系，这将包括对用可计算数表示的实变函数理论的研究。

图灵并没有严格按这种方法深入研究这些主题。你在第17章会看到，当斯蒂芬·克莱尼（在他于1952年出版的《数学引论》中）和马丁·戴维斯（在他于1958年出版的《可计算性与不可解性》中）重新设计了图灵机并用它们来计算整数函数而不是实数时，可计算函数的概念就变得十分重要了。

在某种意义上，我们已经看到了执行函数的机器。通用机就是这样的，因为它把机器的标准描述作为输入，输出中包括该机器的完全格局以及机器本应计算的序列。

更传统的函数呢？它们是什么样子的？想想三角正弦函数。输入是一个表示角度的数值，通常以度或弧度为单位。假定这个角是直角三角形中的一个角，则正弦函数计算该角的对边与斜边的比。更一般地说（定义大于90°的角的正弦函数），从笛卡儿坐标系的原点出发到任意一点画一条线。这条线与X轴的夹角（按逆时针方向测量）的正弦函数值，就是这条线的端点到X轴的距离与该直线长度的比值。

尽管正弦函数是以360°或2π个弧度为周期循环的，但是该函数的定义域由所有实数组成，它的值域（函数的值）由从-1到1之间的实数（包括-1和1）组成。

实际计算正弦函数是要用到如下的无穷级数，其中*x*以弧度为单位：

$$\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\cdots$$

这就是如今的计算机计算正弦函数时所用的公式，尽管一般情况下，这种计算是在计算机的处理芯片而非软件中进行的。

因为实数拥有无穷数目的十进制位，因此计算机不能存储任意的实数，而是用有理数来近似实数。1985年，美国电气和电子工程师协会（IEEE）公布了二进制浮点运算的IEEE标准，很多计算机系统都使用该标准，以一种适合于科学计数法表示的形式存储数字。例如，常用的双精度格式使用64位存储一个数：1位用来存储符号（正或负），11位用来存储指数，52位用来存储尾数，提供大致相当于16位十进制数的精度。[1]实质上，我们是用两个整数来存储实数123.456的，它们分别是8 687 443 681 197 687（尾数）和46（指数），因为8 687 443 681 197 687除以2^{46}近似等于123.456。这个比值是一个有理数，而不是实数。

当计算机计算一个正弦值时，这一近似是很重要的，因为它可以保证函数在有限时间内完成。尽管正弦函数是作为一个无穷级数来计算的，但是这个级数项的绝对值越来越小。当项对结果的精度不再产生影响时，函数就终止了。

然而，图灵机追求无限的精度。计算实数时，它会一直不停地运算。由于计算正弦函数的无限级数中的每个项都包含无限的位数，因此在计算正弦函数时，追求无限的精度是个大问题。例如，当角度是1弧度时，第二项是1/6，无论表示成十进制还是二进制，它都需要无限位数。如果机器需要计算第二项的无限个数位，它怎么样才能运行到对第

三项的计算呢？

一种可行的方法是，依次计算每一项的第一位，直到某一项的第一位数字为0，然后，再依次计算每一项的第二位，直到某一项的前两位数字均为0，依此类推。这显然是一个很复杂的过程，特别是你不想机器在计算得到结果后再擦除结果的任意位时。

执行正弦函数只是一个问题，输入从哪里来呢？

或许我们的直觉是让机器的使用者以某种方式“键入”机器需要计算的角。这显然是受现在的交互式计算机和屏幕计算器的启发，但是为了接受这种形式的输入，需要重新设计图灵机。这比目前我们所做的工作量更大。

第二种观点是将函数的输入“硬编码”在机器内部。例如，我们可以设计一台专门计算37.85°的正弦值的机器。尽管这样会把机器限制为只能求解某一个角度的正弦值，但是我们还是希望设计出的这种机器易于修改，从而可以计算其他角度的正弦值。

第三种方法是把角度编码到纸带上。机器读取这个输入，计算正弦值，然后再把结果打印到纸带上。（我猜你喜欢这么做！我也是。）

第四种方法是让机器自己产生输入。例如，机器首先计算角度为0°的正弦值，然后计算角度为1°的正弦值，再计算角度为2°的正弦值，等等，并把每个结果都打印在纸带上，最后会得到一张包含很多角度正弦值的“表”。这种方法要求机器计算得到的每个结果都只包含有限个数位。

第五种方法需要两台不同的机器。第一台机器计算实数，第二台计算该数的正弦值。我说两台机器的时候，实际上是指可以实现两台机器逻辑的一台机器。我们已经遇到过以这种方式结合的机器。在第8节中（本书第166～167页），图灵把一台判定机器$\mathscr{D}$和通用机$\mathscr{U}$结合起来，构造成机器$\mathscr{H}$来分析标准描述。这种做法的好处是，我们可以“插入”不同的第一台机器来计算不同角度的正弦值。

这些做法都是有问题的。一个大问题是正弦函数的输入和输出都是实数（至少理论上是这样的），而实数包含无限位。键入一个有无限位的数或将这样的数编码在纸带上都是不可能的。

事实上，即使你将角度限制在简单的、可以表示成有限的十进制数的范围内，正弦函数需要的也是弧度制的角度。180°对应对π个弧度，因此看上去很简单的10°其实是其π/18个弧度——一个包含无限个十进制位数的超越数。

现在我们面临的是一台需要计算现π/18弧度的机器和另一台计算该弧度的正弦值的机器，事实上这两台机器是在同一个“元机器”中运行的。第二台机器不可能等到第一台机器计算完成后再开始计算！这两台机器需要以串联模式工作，也就是有时称作“燕尾连接”的一种编程技术。计算角的机器得到一个新的数位时，计算该角度正弦值的机器必须接收这个数位，并计算出结果的一个新的数位。这两台机器间会永远进行这一相互作用的来回。

现在，你可能就不会对斯蒂芬·克莱尼和马丁·戴维斯重新改造的图灵机只能计算数论函数，也就是域和值域都限制在非负整数的函数，感到惊讶了。他们都把函数输入编码成笔画。例如，在8个连续的格中画简单的竖线来表示数字7（数字0只需要1画）。

通常情况下，当计算一个函数时，你不会想让这个计算永久进行下去。你希望函数会结束，这样便可以检测结果。因此，经克莱尼和戴维斯改造的图灵机在计算结束的时候会停止运行。显然，用于计算非负整数的加法和乘法的机器不需要永久地运行。在克莱尼和戴维斯的设计中，不能停止运行的机器被看作是故障机器。戴维斯把判定一台图灵机是否能够正确地完成计算并停机的过程称为停机问题。后来，停机问题被等同于图灵机，但是这个概念与图灵最初的论文无关。

既然我们已经考虑了构造处理实数函数的图灵机时遇到的一些与生俱来的问题，接下来我们准备学习图灵在他论文的第10节中提出的解决方法。

第10节其实是第9节的延续。前一节的开头，图灵对于其机器的计算能力问题给出了三个论据。第三个是第10节的主题，即“给出一大类可计算数的示例”。同时，图灵讨论的重点稍有改变，从原来的可计算数转移到了可计算函数的范围。

第10节可能是图灵论文中最少被分析的部分，通常也是最难理解的一节。他写得很简洁，有时还有点模糊。我没有信心总是能够完全准确地揭示他的论点。

也许这并不令人惊讶，图灵一开始就讨论了“带有整数变量的可计算函数”，并且定义这样函数的“最简单的”方式需要非负整数的定义域和值域。

10. *Examples of large classes of numbers which are computable.*

It will be useful to begin with definitions of a computable function of an integral variable and of a computable variable, etc. There are many equivalent ways of defining a computable function of an integral

variable. The simplest is, possibly, as follows. If γ is a computable sequence in which 0 appears infinitely[†] often, and n is an integer, then let us define $\xi(\gamma,n)$ to be the number of figures 1 between the n-th and the $(n + 1)$-th figure 0 in γ. Then $\phi(n)$ is computable if, for all n and come γ, $\phi(n) = \xi(\gamma,n)$.

† If $\mathcal{M}$ computes γ, then the problem whether $\mathcal{M}$ prints 0 infinitely often is of the same character as the problem whether $\mathcal{M}$ is circle-free.

10. 大量可计算数的示例

以定义整数变量和可计算变量的可计算函数开始，会非常有用。有很多等价的方法用来定义整数变量的可计算函数。下面的定义可能是最简单的。如果γ是一个可计算序列，其中0出现无穷次[†]，n是一个整数，我们定义$\xi(\gamma, n)$为γ中第n个0和第$(n+1)$个0之间数字1的个数。如果对于所有的n，存在某个γ，使得$\phi(n) = \xi(\gamma, n)$，则$\phi(n)$是可计算的。

† 如果$\mathcal{M}$计算γ，则$\mathcal{M}$是否无限打印0的问题与$\mathcal{M}$是否是非循环的问题是同一性质。

我需要举例子说明。假设函数ϕ（希腊字母phi）简单定义为：

$$\phi(n) = 2n + 1,$$

其中n为非负整数。则，

$$\phi(0) = 1$$
$$\phi(1) = 3$$
$$\phi(2) = 5$$
$$\phi(3) = 7$$
$$\ldots$$

对应于该函数的序列γ（希腊字母gamma）为：

$$010111011111011111110\ldots$$

注意，两个相邻的0之间分别是一个1、三个1、五个1、七个1等，1的数目与每个连续的非负整数n的函数$\phi(n)$的值相对应。

γ是可计算序列吗？对我而言，它的确看上去是可计算的。这意味着$\phi(n)$是一个可计算函数，计算γ的机器永远不停地运行，计算出函数所有的值。

图灵从与我的例子完全相反的角度处理这个可计算函数的问题：他假定γ序列总是包含无穷个0，函数 $\xi(\gamma, n)$（希腊字母xi）表示每对相邻的0之间数字1的个数，并且 $\phi(n)=\xi(\gamma, n)$。

从这点看，计算包含无穷个0的序列的任何机器也是在计算一个正整数值的函数，尽管一般情况下，我们不能判定机器是否确实满足这个标准。

现在，图灵假定一个与函数ϕ相对应的谓词，使得该函数的计算类似于数字的基于逻辑的演算，也就是图灵在论文第9节、本书前一章中论述的内容。

An equivalent definition is this. Let $H(x, y)$ mean $\phi(x)=y$.

如此，我们得到一种等价的定义。定义$H(x, y)$，其中$\phi(x) = y$。

我们使用相同的例子。改变函数的自变量，现在的定义如下：

$$\phi(x) = 2x+1$$

使用前一章中描述的谓词I来定义$H(x, y)$：

$$(\exists z)(\mathrm{Product}(u'', x, z) \;\&\; \mathrm{Sum}(z, u', y))$$

如果存在某个z，使得2乘以x等于z，且z加1等于y均为真，则该公式为真。

Then, if we can find a contradiction-free axioḿ $\mathfrak{A}_\phi$, such that $\mathfrak{A}$ $\varphi \to P$,

然后，如果我们能找到一个非矛盾的公理$\mathfrak{A}_{ø}$，使得$\mathfrak{A}_{ø}$ → *P*，

则$\mathfrak{A}_{ø}$一定是定义*H*所需谓词的公理（这个例子中的Product和Sum）与*P*的合取式，因此这个蕴涵式很容易就成立。

and if for each integer *n* there exists an integer *N*, such that

$$\mathfrak{A}_\phi \ \& \ F^{(N)} \rightarrow H(u^{(n)}, u^{(\phi(n))}),$$

并且如果对每个整数*n*都存在一个整数*N*，使得

$$\mathfrak{A}_\phi \ \& \ F^{(N)} \rightarrow H(u^{(n)}, u^{(\phi(n))}),$$

你会想起，$F^{(n)}$是后继函数的合取式的缩写形式。*N*必须至少等于*n*和*ø*(*n*)中较大的一个。在我们的例子中，当*n* = 10时，*ø*(*n*)=21。函数*H*内部需要数字1和2，并且*z*的值为20。因此，*N*必须至少等于21才能定义足够多的数字，而在某些情况下，它可以比*n*和*ø*(*n*)的值都大。

and such that, if *m*≠*ø*(*n*), then, for some *N*′,

$$\mathfrak{A}_\phi \ \& \ F^{(N')} \rightarrow (—H(u^{(n)}, u^{(m)}),$$

且使得如果*m*≠*ø*(*n*)，则对某个*N’*，

$$\mathfrak{A}_\phi \ \& \ F^{(N')} \rightarrow (—H(u^{(n)}, u^{(m)}),$$

公式末尾漏掉了右半括号。总之，当第一个参数为*n*，第二个参数

为*ø*(*n*)时，谓词*H*为真，否则为假。对于第二个公式，*N'*的值也至少要等于*m*，但是稍后你将看到，我们在演示中不会涉及比*n*大的*m*值。

then ø may be said to be a computable function.

则可以说*ø*是一个可计算函数。

到这儿，图灵结束了他的讨论，但并没有真正描述它应该如何发挥作用。机器 $\mathcal{K}$ 的另一种修正，是枚举出所有可证明的公式。图灵在第9节描述的机器针对*n*的连续取值依次枚举出这些可证明的公式，根据谓词*G*的真值，为*n*的每个值打印一个0或1。

这里的新问题需要一台机器 $\mathcal{K}_\gamma$来计算前面描述的序列*γ*：

010111011110111111110…

其中，每个1串中数字“1”的数目为对应于*n*的连续值的函数*ø*(*n*)的值。最大的不同是，这台机器不仅对连续的*n*值，还对变化的*n*和*m*的值，枚举出所有可证明的公式。

对于*n*和*m*的每个新值，机器首先打印公式A和B（仍使用为之前的机器设置的术语）：

$$\mathbf{A}\ \mathfrak{A}_\phi \ \&\ F^{(N)} \rightarrow H(u^{(n)}, u^{(m)}) \quad \mathbf{B}\ \mathfrak{A}_\phi \ \&\ F^{(N')} \rightarrow \left(-H(u^{(n)}, u^{(m)})\right)$$

然后通过产生所有的可证明的公式，尝试与二者中的一个进行匹配。

机器从*n*等于0开始计算。*n*是函数*ø*(*n*)的参数，也是谓词*H*(*n, m*)的第一个参数。对*n*的每个新的值，机器打印一个0，然后把*m*设置为0。*m*可能是函数*ø*(*n*)的结果，它是谓词*H*(*n, m*)的第二个参数。

对*m*的每个新的取值，机器开始产生所有可证明的公式。如果与公式B匹配，则机器打印一个1，因为*m*的值不是函数*ø*(*n*)的结果。机器增大*m*，打印新的A和B，然后再次开始产生可证明的公式。如果与公式A匹配，则该*m*值是函数*ø*(*n*)的结果，机器继续对*n*的下一个值进行处理。对于*n*的下一个值，机器开始打印一个0，并且将*m*设置回0。

以这种方式，机器打印出序列*γ*，其中的每个1串中数字1的数目都表示依次增大的整数参数对应的函数值。

We cannot define general computable functions of a real variable, since there is no general method of describing a real number,

由于没有描述实数的一般方法，因而我们无法定义一般的实数变量的可计算函数。

这就是我之前考虑将实数编码在纸带上供函数处理时遇到的问题。

but we can define a computable function of a computable variable.

但是，我们可以定义一个以可计算数为变量的可计算函数。

这需要可计算数的计算和可计算函数的计算串联工作。在或许是最简单的情况下，每次计算得到变量的一个新数位后，可计算函数就会接收它，并打印出结果的新的数位。可计算变量和可计算函数始终保持相同的有效数位，从而保持相同的精度。

图灵所采用的例子是三角正切函数。他想对各种各样的，其实是所有的可计算数计算它们的正切值，但是他没有为一个计算的终止和另一个计算的启动指定任何标准。

If n is satisfactory, let γ_n be the number computed by $\mathcal{M}(n)$, and let

$$\alpha_n = \tan\left(\pi\left(\gamma_n - \frac{1}{2}\right)\right),$$

[255]

unless $\gamma_n = 0$ or $\gamma_n = 1$, in either of which cases $\alpha_n = 0$.

如果n是可接受的，假定γ_n为机器$\mathcal{M}$(n)计算得到的数字，并且

$$\alpha_n = \tan(\pi(\gamma_n - 1/2))$$

除非γ_n＝0或者γ_n＝1。在这两种情况下，α_n＝0。

后面的脚注表明这不是唯一可能的函数，不过图灵特地挑选了一个简单的例子。因为图灵机计算0和1之间的实数，所以无论机器如何定义，γ_n的值都介于0和1之间。正切函数的参数范围介于之1/2π和1/2π之间，这是用弧度制表示的角，等价的角度范围是之-90°到90°。

Then, as *n* runs through the satisfactory numbers, α_n runs through the computable numbers†.

† A function α_n may be defined in many other ways so as to run through the computable numbers.

接着，随着n遍历可接受数，α_n遍历可计算数†。

† 函数α_n也可以用许多其他方式来定义，从而能够遍历可计算数。

-90°到90°之间角度的正切值的变化范围是（–∞, ∞），因此它覆盖了整个实数集。（严格地说，）-90°和90°角的正切值没有定义，这也正是为什么图灵单独处理这两种情况的原因。）除去极少数例外，正切函数的结果都是超越数。

图灵后来才建议，我们事实上可以定义计算一个角的正切值的图灵机。跟正弦函数相似，正切函数也可以通过一个无穷级数来计算：

$$\tan(x) = x + \frac{x^3}{3} + \frac{x^5}{3 \cdot 5} + \frac{x^7}{3 \cdot 5 \cdot 7} + \cdots$$

x的变化范围从围1/2π到1/2π。

Now let $ø(n)$ be a computable function which can be shown to be such that for any satisfactory argument its value is satisfactory[‡]. Then the function *f*, defined by $f(\alpha_n)=\alpha_{ø(n)}$, is a computable function and all computable functions of a computable variable are expressible in this form.

[‡] Although it is not possible to find a general process for determining whether a given number is satisfactory, it is often possible to show that certain classes of numbers are satisfactory.

令$ø(n)$为一个可计算函数，对任意的可接受参数，它的值都是可接受的[†]。则定义为$f(\alpha_n)=\alpha_ø(n)$的函数*f*是一个可计算函数，并且所有可计算变量的可计算函数都可以表示成这种形式。

[†] 尽管不可能找到可以判定某个给定的数字是否是可接受的通用过程，但是通常情况下，表明某一特定类的数字是可接受的还是有可能的。

函数$ø(n)$的定义域和值域都是符合要求的机器的描述数。或许我们只考虑某种特定格式的机器，很容易就可以将其构造成是符合要求的。函数$ø(n)$修改描述数，其实是重新编程机器，使得它可以计算出其他东西。例如，$ø(n)$可以重新编程机器，使它不再求解x，而是x的两倍再加1，即执行函数$2x + 1$。

Similar definitions may be given of computable functions of several variables, computable-valued functions of an integral variable, etc.

I shall enunciate a number of theorems about computability, but I shall prove only (ii) and a theorem similar to (iii).

类似地，也可以定义包含几个变量的可计算函数、包含整数变量且值为可计算数的函数，等等。

我将给出一些关于可计算性的定理，但是只证明(ii)及与(iii)相似的定理。

随后的十个定理用小写罗马数字编号。图灵论文第10节的最后两页给出了对(ii)和(iii)的证明。

> (i) A computable function of a computable function of an integral or computable variable is computable.

> (i) 以整数或可计算数为变量的可计算函数的可计算函数是可计算的。

换句话说，我们可以把这些东西堆积起来。从计算一个数的机器出发，使用另一台机器执行这个数的函数，然后再使用一台机器执行第一个函数的结果的函数。与之前基于三角正切函数的机器类似，这些机器不会等待前一阶段完成以后再运行。它们必须以串联方式工作，或许是随着每一个数位的计算，从一个阶段向另一个阶段传递信息。

> (ii) Any function of an integral variable defined recursively in terms of computable functions is cpmputable. *I.e.* if ø(*m*,*n*) is computable, and *r* is some integer, then *η*(*n*) is computable, where

$$\eta(0) = r,$$

$$\eta(n) = \phi\left(n, \eta(n-1)\right).$$

> (ii)以可计算函数递归定义的任何包含整数变量的函数是可计算的。例如，如果ø(*m*, *n*)是可计算的，*r*是整数，则*η*(*n*)是可计算的，其中，

$$\eta(0) = r,$$

$$\eta(n) = \phi\,(n, \eta(n-1))$$

注意，希腊字母*η*（读为eta）与斜体的*n*有点相像。“包含整数变量的函数”是*η*，“可计算函数”指的是ø(*m*, *n*)。作为例子，我们假设*r* = 1，函数ø(*m*, *n*)简单定义为：

$$\phi(m, n) = m \cdot n$$

我确定$ø(m, n)$是可计算的。我们计算η的几个值：

$\eta(0) = r = 1$

$\eta(1) = ø(1, \eta(0)) = ø(1, 1) = 1·1 = 1$

$\eta(2) = ø(2, \eta(1)) = ø(2, 1) = 2·1 = 2$

$\eta(3) = ø(3, \eta(2)) = ø(3, 2) = 3·2 = 6$

$\eta(4) = ø(4, \eta(3)) = ø(4, 6) = 4·6 = 24$

$\eta(5) = ø(5, \eta(4)) = ø(5, 24) = 5·24 = 120$

…

函数$\eta(n)$是阶乘函数的递归定义。到本节最后，图灵将证明使用可计算函数$ø(m, n)$定义的整数函数（如 $\eta(n)$）本身也是可计算的。

(iii) If $ø(m, n)$ is a computable function of two integral variables, then $ø(n, n)$ is a computable function of n.

(iii) 如果 $ø(m, n)$是一个包含两个整数变量的可计算函数，则$ø(n, n)$是关于的可计算函数。

这个定理听起来很平凡，但图灵利用这个机会做了一些有趣的事。本章以对它的证明结束。

(iv) If $ø(n)$ is a computable function whose value is always 0 or 1, then the sequence whose n-th figure is $ø(n)$ is computable.

(iv) 如果$ø(n)$是一个其值总为0或1的可计算函数，那么第n位数字为$ø(n)$的序列是可计算的。

例如，假设函数IsPrime(n)当n是素数时返回1，否则返回0。图灵断定下面的序列是可计算的：

0011010100010100010100010000010…

我只展示了n从0开始的前31位数字。与素数2、3、5、7、11、13、17、19、23和29对应的数位都设置成1。

图灵接着引用了戴德金定理。这里给出了G. H. 哈代的《纯数学教程》第1章中对该定理的陈述。1931年，图灵为准备剑桥大学的秋季入学考试开始阅读这本书，那也可能是他第一次遇到这个定理。

> 戴德金定理 如果按照以下的方法把实数集合分割成两类L和R：
>
> (i) 每个数只属于两个类中的一类；
>
> (ii) 每个类至少包含一个数；
>
> (iii) L中的任意一个数都小于R中的任意一个数。
>
> 则存在一个数α，满足小于它的所有数都属于L类，大于它的所有数都属于R类。数α本身可能属于L或R中的任意一类。[2]

第一次看到这个定理时容易感到迷惑，但是它描述了有理数和实数之间最基本的区别，特别是，实数是如何组成了连续统，而有理数不能。

想象从左边的负无穷到右边的正无穷有一条数轴，也可以只考虑这个数轴的一段。把它分割成两个部分（这称为戴德金分割）。一些数在左边的线上（定理中的L），另一些数在右边的线上（R）。

例如，你可以在1.5处分割这条直线，因此L中的所有数都小于1.5，R中的所有数都大于1.5。那1.5自身呢？你可以把它放在L中，也可以放在R中。把1.5放在L中，它是L中的最大数，R中没有最小数。换句话说，R中没有一个比其余的数字都小的数。另一方面，把1.5放在R中，它是R中的最小数，L中没有最大数。

在2的平方根处分割这条直线。如果这条数轴仅由有理数组成，那么L中的所有数都小于2的平方根，R中的所有数都大于2的平方根。因为2的平方根不是有理数，它既不属于L，也不属于R。此外，L中没有最大数，R中也没有最小数。这条直线在2的平方根处出现间断。

然而，如果这条数轴是由实数组成的，则2的平方根必须属于L或者R。你无法定义一个实数的分割，使得这个分割点既不属于L也不属于R。这就是为什么实数可以组成连续统，而有理数不能的原因。

> Dedekind's theorem does not hold in the ordinary form if we replace “real” throughout by “computable”.

如果我们用“可计算数”替换“实数”，那么一般形式表示的戴德

金定理就不成立了。

可计算数不能组成连续统，因为你可以在一个非可计算数处做戴德金分割。这个数也许太复杂（也就是太随机）而难以用算法定义，或者你也可以定义这个数，例如，由每个可计算数中的一个数位构成的数，但无法计算它。这个分割把可计算数分成两个部分，使得L中没有最大数，R中也没有最小数。

But it holds in the following form:

但是用下面的形式表示，戴德金定理仍然成立：

注意下面的罗马数字，这是在图灵定理(v)提出的。当参数（一个可计算数）小于（或者小于等于）某个固定数ξ（希腊字母xi）时，图灵在这里引入的命题函数$G(\alpha)$为真。

函数$G(\alpha)$把可计算数分成两类：L，使$G(\alpha)$返回真的可计算数；R，使$G(\alpha)$返回假的可计算数。哈代关于戴德金定理的描述满足条件(i)，因为对于每个可计算数，$G(\alpha)$的值或为真或为假。

(v) If $G(\alpha)$ is a propositional function of the computable numbers and

$$(a)\quad (\exists\alpha)(\exists\beta)\left\{G(\alpha)\ \&\ \left(-G(\beta)\right)\right\},$$

(v) 如果$G(\alpha)$是可计算数的命题函数，并且

$$(a)\quad (\exists\alpha)(\exists\beta)\left\{G(\alpha)\ \&\ \left(-G(\beta)\right)\right\},$$

图灵的公式(a)与哈代的条件(ii)等价，也就是每一类至少包含一个元素。

$$(b)\quad G(\alpha)\ \&\ \big(—G(\beta)\big) \rightarrow (\alpha < \beta),$$

$$(b)\quad G(\alpha)\ \&\ \big(—G(\beta)\big) \rightarrow (\alpha < \beta),$$

图灵的公式(b)与哈代的条件(iii)等价。如果$G(\alpha)$为真，那么α在L中；如果$G(\beta)$为假，则β在R中，且α小于β。

and there is a geral process for determining the truth value of $G(\alpha)$, then [256] there is a computable number ξ such that

[256]

there is a computable number ξ such that

$$G(\alpha) \rightarrow \alpha \leq \xi,$$
$$—G(\alpha) \rightarrow \alpha \geq \xi.$$

并且存在判定$G(\alpha)$真值的通用过程，那么存在一个可计算数 ξ 满足：

$$G(\alpha) \rightarrow \alpha \leq \xi,$$
$$—G(\alpha) \rightarrow \alpha \geq \xi.$$

“判定$G(\alpha)$真值的通用过程”的条件很重要。因为ξ是一个可计算数，一般情况下，它不能显式地存储在$G(\alpha)$的定义中。然而，通过不断缩小这个值的范围，机器是有可能计算得到该数的。事实上，可以不断地检测$G(\alpha)$的更加准确的值，计算ξ的机器就是这么做的。

In other words, the theorem holds for any section of the computables such that there is a general process for determining to which class a given number belongs.

换句话说，对于任意的可计算数，只要存在判定某个给定的数属于哪一类的通用过程，戴德金定理都是成立的。

在下一句话中，“可计算数序列”这个表达可能有点模糊，因为几乎从论文的开始，图灵都是使用“可计算序列”来指代机器产生的数位。他这里所说的“序列”指的是可计算数的有序集合。

Owing to this restriction of Dedekind's theorem, we cannot say that a computable bounded increasing sequence of computable numbers has a computable limit. This may possibly be understood by considering a sequence such as

$$-1, -\frac{1}{2}, -\frac{1}{4}, -\frac{1}{8}, -\frac{1}{16}, \frac{1}{2}, \ \ldots .$$

由于戴德金定理的这个局限性，我们不能声称一个可计算的有界递增可计算数序列是有可计算的极限的。看看下面的序列或许有助理解：

$$-1, -\frac{1}{2}, -\frac{1}{4}, -\frac{1}{8}, -\frac{1}{16}, \frac{1}{2}, \ \ldots .$$

几年前，sci.logic新闻组的讨论主题就是这一小段文字，http://sci.techarchive. net/Archive/sci.logic/2004-08/2244.html上记录了这一进程。一种见解是，最后的1/2是错误的，事实上应该是是1/32。修正后，这个序列好看多了，但是它不可能是图灵所想的，因为如果是这样，这个序列显然越来越接近于0，而0当然是一个可计算的极限。

一种貌似更加合理的推测是，图灵正在展示的序列看上去是趋向一个可计算的极限，但事实上并非如此。这个序列或许约束在一1和1之间，但是它并没有告诉我们这个极限是否真的可计算。

On the other hand, (v)enables us to prove

(vi) If α and β are computable and $\alpha < \beta$ and $ø(\alpha) < 0 < ø(\beta)$, where $ø(\alpha)$ is a computable increasing continuous function, then there is a unique computable number γ, satisfying $\alpha < \gamma < \beta$ and $ø(\gamma) = 0$.

另一方面，(v)使得我们可以证明：

(vi)如果α和β是可计算的，且$\alpha<\beta$，$ø(\alpha)<0<ø(\beta)$，其中，$ø(\alpha)$是一个可计算的递增连续函数，那么存在唯一的可计算数γ满足$\alpha<\gamma<\beta$，且$ø(\gamma)=0$。

图灵暗指一个可计算函数，它可能是一个多项式：

$$\phi(x) = 5x^3 - 3x^2 - 7$$

众所周知，任何奇数次（最大的指数）的实多项式都至少有一个实根，也就是说，有一个值x使得这个多项式的值为0。

对于多项式$ø(x)$，通过观察$ø(1)=-5$，$ø(2)=21$，可以缩小根的范围。这个根必须介于1和2之间。

数字1和2就是图灵描述中的α和β，你可以发现$\alpha < \beta$且$ø(\alpha) <0<(\beta)$。因为这个函数是连续的（即没有中断），所以存在一个值γ，它介于α和β之间，且$ø(\gamma)$的值为0。我们目标就是计算这个γ。我们可以在α和β的中间选择一个数，如1.5，计算$ø(1.5)$，结果为3.125。因为3.125大于0，所以可以把1.5作为新的β值。现在我们知道根是1和1.5之间的某个数。再尝试1.25。$ø(1.25) = (1.921875$，结果小于0，因此把1.25作为新的α值。现在我们知道这个根介于1.25和1.5之间。每一步都可以把这个根约束在一个更小的范围内，事实上就是在计算这个根。

粗略地说，如果一个数列中两个连续数的差的绝对值越来越小，那么我们称这个数列是收敛的。（我说“粗略地”是因为在序列的前面部分，这些差值的绝对值可能不会变小。）任何实数都可以表示成一个有理数的收敛序列。最简单地，可以使用如下的有理数序列来描述一个实数：

$\alpha_0 = 3$

$\alpha_1 = 3.1$
$\alpha_2 = 3.14$
$\alpha_3 = 3,141$
$\alpha_4 = 3,1415$
$\alpha_5 = 3/14159$
…
这些全是有理数，它们都越来越接近于无理数这。这个序列是收敛的，因为连续的两个数的差值越来越小，如a_3和a_4的差为0.0005，而a_4和a_5的差为0.00009。

数学上，常把这个差用一个小写的ε（读为epsilon）表示。你可以选择任意小的ε值，如0.0001。对于这个ε值，如果存在某个数N，使得对所有的$n > N$和$m > N$，对应项a_n和a_m的差的绝对值都小于我们所选的任意小的数，即$\left|a_n - a_m\right| < \varepsilon$，则该序列是收敛的。在上面的例子中，$\varepsilon$的值为0.0001，$N$为3，因为$\left|a_4 - a_5\right| < 0.0001$，当$n$和$m$大于3时，$a_n$和$a_m$的差也同样会小于0.0001。

图灵以类似的方式定义了可计算收敛。

Computable convergence.

We shall say that a sequence β_n of computable numbers *converges computably* if there is a computable integral valued function $N(\eta)$ of the computable variable η, such that we can show that, if $\eta > 0$ and $n > N(\eta)$ and $m > N(\eta)$, then $|\beta_n - \beta_m| < \epsilon.$

可计算收敛

如果对于可计算变量ε存在一个可计算整数函数$N(\varepsilon)$，使得对于任意$\varepsilon>0$，$n>N(\varepsilon)$，$m>N(\varepsilon)$，则$\left|\beta_n - \beta_m\right| < \varepsilon$，那么我们称这个由可计算数组成的序列$\beta_n$为可计算收敛的。

在这里，序列中的数必须是可计算的，图灵还要求ε是可计算的，且$N(\varepsilon)$是可计算函数。

We can then show that

(vii) A power series whose coefficients form a computable sequence of computable numbers is computably convergent at all computable points in the interior of its interval of convergence.

我们可以证明：

(vii) 如果一个幂级数的系数构成了一个可计算数的可计算序列，那么这个幂级数在其收敛域内的可计算点处是可计算收敛的。

一个幂级数是形如：

$$a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots$$

的无限和。正如我之前介绍的，你可以把三角正弦函数用幂级数形式表示出来：

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

这些系数（即a_i的值）是1、0、-1/3!、0、1/5!、0、1/7!，等等。这些系数当然是一个可计算序列。某些幂级数只有在x等于0时才收敛。另一些则在x的某个值域范围收敛，这个范围称为收敛域。众所周知，正弦函数对于任何x都是收敛的。因为这些系数都是可计算的，所以用机器来判定它们是否收敛是可能的。

(viii) The limit of a computably convergent sequencee is computable.

(viii) 可计算收敛序列的极限是可计算的。

也可能存在一组函数，收敛到某个特定的函数。如果这个收敛是在函数的某个特定值处出现的，称作逐点收敛。更强的函数收敛又称为一致收敛，这种收敛并不依赖于函数的取值。

And with the obvious definition of “uniformly computably convergent”:

(ix) The limit of a uniformly computably convergent computable sequence of computable functions is a computable function. Hence

(x) The sum of a power series whose coefficients form a computable sequence is a computable function in the interior of its interval of convergence.

根据“一致可计算收敛”的定义，显然有：

(ix) 由可计算函数组成的一致可计算收敛的可计算序列的极限是一个可计算函数，因此

(x) 一个其系数形成可计算序列的幂级数的和是在其收敛域内的可计算函数。

根据这些定理，图灵总结出所有的代数数和一些常见的超越数都是可计算的。

From (vii) and $\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \ldots)$ we deduce that π iscomputable.

From $e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \ldots$ We deduce that *e* is computable.

[257]

From (vi) we deduce that all real algebraic numbers are computable.

从(viii)和 $\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \cdots)$, 我们可推出π是可计算的。

从 $e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots$ 可以推出e是可计算的。

从(vi)中，可以推出所有的实代数数都是可计算的。

回想一下，代数数是多项式方程的根。

From (vi) and (x) we deduce that the real zeros of the Bessel functions are computable.

从(vi)和(x)中，我们推断出贝塞尔函数的实零点是可计算的。

贝塞尔函数是微分方程的一种常见形式的解。零点就是这些函数取0值的点。最后的结论也包括了三角函数、对数函数、指数函数和一些不太知名的函数。

图灵之前说要证明定理(ii)，该定理断言任何整数变量的以可计算函数递归定义的函数是可计算的。

Proof of (ii).

(ii)的证明。

在第10节的开始，图灵定义了一个谓词$H(x, y)$，如果$ø(x) = y$，则$H(x, y)$为真。而且介绍了当$m \neq ø(n)$时，机器如何证明包含 $H(u^{(n)}, u^{(\phi(n))})$ 和 $-H(u^{(n)}, u^{(m)})$ 的公式。那个证明建立了函数$ø(x)$的可计算性。

当前的证明基于早先的证明，但是这里他用$\eta(x)$替换了$ø(x)$，

其中：

$$\eta(0) = r$$
$$\eta(n) = \phi(n, \eta(n-1))$$

Let $H(x, y)$ mean "$\eta(x)=y$", and let $K(x, y, z)$ mean "$ø(x, y)=z$".

令$H(x, y)$表示$\eta(x) = y$，并且令$K(x, y, z)$表示$ø(x, y) = z$。

对于阶乘的例子：

$$\phi(x, y) = x \cdot y$$

因此$K(x, y, z)$就是Product(x, y, z)。

$\mathfrak{A}_{ø}$ is the axiom for $ø(x, y)$.

$\mathfrak{A}_{ø}$是$ø(x, y)$的公理。

这个$\mathfrak{A}_{\varphi}$公理需要提供对谓词$K(x, y, z)$的支持。对于阶乘的例子，这个公理包括对后继、谓词Sum与Product的公理。下面$\eta(x)$的公理更加详尽。

We that $\mathfrak{A}_{\eta}$ to be

$$\mathfrak{A}_\phi \,\&\, P \,\&\, \big(F(x,y) \to G(x,y)\big) \,\&\, \big(G(x,y) \,\&\, G(y,z) \to G(x,z)\big)$$
$$\&\, \big(F^{(r)} \to H(u,u^{(r)})\big) \,\&\, \big(F(v,w) \,\&\, H(v,x) \,\&\, K(w,x,z) \to H(w,z)\big)$$
$$\&\, \big[H(w,z) \,\&\, G(z,t) \vee G(t,z) \to \big(-H(w,t)\big)\big].$$

我们将 $\mathfrak{A}_\eta$定义为：

$$\mathfrak{A}_\phi \,\&\, P \,\&\, \big(F(x,y) \to G(x,y)\big) \,\&\, \big(G(x,y) \,\&\, G(y,z) \to G(x,z)\big)$$
$$\&\, \big(F^{(r)} \to H(u,u^{(r)})\big) \,\&\, \big(F(v,w) \,\&\, H(v,x) \,\&\, K(w,x,z) \to H(w,z)\big)$$
$$\&\, \big[H(w,z) \,\&\, G(z,t) \vee G(t,z) \to \big(-H(w,t)\big)\big].$$

这是图灵第三次使用名为*G*的谓词，它们有着不同的定义。然而，这一次的含义最基础：它就是“大于”函数。奇怪的是，图灵并没有澄清这个函数的功能，直到《伦敦数学学会集刊》（*Proceedings of the London Mathematical Society*）发表了图灵这篇论文的修订稿[3]时才予以说明。（这篇修订稿在本书的第16章）。在修订的论文中，图灵声称$G(x, y)$可以阐述为“x在y之前”或$y > x$。像往常一样，*F*谓词就是后继函数。

$\mathfrak{A}_\eta$是7个表达式的合取，以$\mathfrak{A}_ø$和*P*开始。第三个表达式表明，如果*y*是*x*的后继，那么*y*大于*x*。第一行的最后一个表达式阐述了*G*函数的传递性（如果*y*大于*x*，*z*大于*y*，那么*z*大于*x*）。

第二行的第一个表达式表明 $H(u, u^{(r)})$为真，这意味着$\eta(0) = r$。第二个表达式是：

$$(F(v,w) \,\&\, H(v,x) \,\&\, K(w,x,z) \to H(w,z))$$

翻译为非谓语的函数就是，如果$w = v + 1$，且$\eta(v) = x$，$ø(w, x) = z$，那么 $\eta(w) = z$, 或者:

$$\eta(v+1) = \phi(v+1, \eta(v))$$

这是另一种阐述*n*大于0时$\eta(n)$的一般公式的方法。

$\mathfrak{A}_\eta$的最后一个表达式是：

$$[H(w,z)\ \&\ G(z,t)\ v\ G(t,z) \rightarrow (-H(w,t))]$$

注意这里的一对*G*谓词，它们的参数互换了。这是将*G*函数并入到此公理的唯一原因。当*z*不等于*t*时，*G*会有一个表达式为真。整个表达式断言，如果$H(w, z)$为真，那么对所有不等于*z*的*t*，$H(w, t)$为假，换言之，$\eta(w) \neq t$。

如果$\mathfrak{A}_\eta$的公式中能够包含许多全称量词，而不是暗示它们的不存在，那么读起来会更加舒服。

I shall not give the proof of consistenc of $\mathfrak{A}_\eta$. Such a proof may be constructed by the methods used in Hilbert and Bernays, *Grundlagen der Mathematik* (Berlin, 1934), p. 209 *et seq*. The consistency is also clear from the meaning.

我不会给出$\mathfrak{A}_\eta$一致性的证明。这些证明可以由希尔伯特和贝奈斯所著《数学基础》（Berlin，1934）一书第209页中所用的方法构造。从这个含义来看，一致性也非常清楚。

这是指大部分内容由瑞士数学家保罗·贝奈斯撰写的第1卷，他在哥廷根时开始写这本书。因为是犹太人，贝奈斯在1933年失去了教授的职称，并于1934年移居到苏黎世。《数学基础》的第2卷在1939年出版。这本书当时备受推崇，但从未翻译为英文版。贝奈斯在图灵的论文中发挥了另一个重要作用。在图灵发表的论文修订稿中，图灵指出他“感激贝奈斯先生指出这些错误”。阿隆佐·邱奇关于判定性问题无解[4]的短论文也受到相同待遇，他同样附上了修

订版[5]，并在脚注上说明“作者感激保罗·贝奈斯先生指出了这个错误”。

尽管贝奈斯从1935~1936年都在高等研究院，但他在图灵抵达普林斯顿前就回到了苏黎世，很显然他们从没见过面。

图灵所引用的页数是在数论那一节的开始处。就是在那一页，贝奈斯引入了R谓词，这与图灵的G谓词相同。贝奈斯为他的大于谓词（这里我将其转换为图灵的记号）列举出了公理，可以看出图灵在多大程度上忽略了这些细节：

$$(x)-R(x,x)$$

$$(x)(y)(z)\big(R(x,y)\,\&\,R(y,z)\rightarrow R(x,z)\big)$$

$$(x)(\exists y)R(x,y)$$

第三个公理提醒我们，存在一个不大于任何数的数。这个数通常是0或1，依赖于人们对于自然数的定义。在《数学基础》的同一页列出了贝奈斯后继函数（称为S）的公理：

$$(x)(\exists y)S(x,y)$$

$$(\exists x)(y)-S(x,y)$$

$$(x)(y)(r)(s)\big(S(x,r)\,\&\,S(y,r)\,\&\,S(s,x)\rightarrow S(s,y)\big)$$

这非常奇怪，因为图灵所引用页定义的谓词正是他在论文中使用的，而他却忽略了这一页上的公理。

图灵在这里用到了归纳法证明，这种证明方法特别适合数论和其他一些只涉及自然数运算的应用。在归纳法证明中，一个公式（或其他的东西）首先被证明在0时满足。这通常很简单。然后，可以假设当数为n时公式为真。再基于此假设，证明在$n + 1$时公式也为真。没必要对每一个n证明公式，只要对n为真就能够蕴涵对$n + 1$也为真。因为公式一开始证明对于0为真，那么对于0 + 1（即1）也为真，而又因为对于1为真，所以对于1+1（即2）也同样成立，依此类推。

图灵的归纳证明有点不同，他把在0点的证明放在了后面，而以归纳的过程作为开始，并指明如果公式对于n-1为真，那么对于n

也为真。他先做了假设。

Suppose that, for some *n*, *N*, we have shown

$$\mathfrak{A}_\eta \ \& \ F^{(N)} \rightarrow H(u^{(n-1)}, u^{(\eta(n-1))}),$$

假设对于某个*n*和*N*，我们有：

$$\mathfrak{A}_\eta \ \& \ F^{(N)} \rightarrow H\left(u^{(n-1)}, u^{(\eta(n-1))}\right),$$

下面这个公式就是$\mathfrak{A}_\phi$公理的直接结果，这个公理支持命题函数*K*。

then, for some *M*,

$$\mathfrak{A}_\phi \ \& \ F^{(M)} \rightarrow K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}),$$

则对于某个*M*，

$$\mathfrak{A}_\phi \ \& \ F^{(M)} \rightarrow K\left(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}\right),$$

同样，因为$\mathfrak{A}_\eta \rightarrow \mathfrak{A}_\phi$很显然，

$$\mathfrak{A}_\eta \ \& \ F^{(M)} \rightarrow K\left(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}\right)$$

如果将我们已知为真的表达式在右侧形成合取式，则上式仍为真。一个是用$F^{(M)}$表示的平凡的后继函数，另一个则是原始的*H*谓

词假设：

$$\mathfrak{A}_\eta \ \& \ F^{(M)} \rightarrow F(u^{(n-1)}, u^{(n)}) \ \& \ H\left(u^{(n-1)}, u^{(\eta(n-1))}\right) \ \& \ K\left(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}\right),$$

$$\mathfrak{A}_\eta \ \& \ F^{(M)} \rightarrow F\left(u^{(n-1)}, u^{(n)}\right) \& \ H\left(u^{(n-1)}, u^{(\eta(n-1))}\right) \ \& K\left(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}\right),$$

这里开始采取$\mathfrak{A}_\eta$公理中的形式，图灵将其抽取了出来：

and

$$\mathfrak{A}_\eta \ \& \ F^{(M)} \rightarrow [F(u^{(n-1)}, u^{(n)}) \ \& \ H(u^{(n-1)}, u^{(\eta(n-1))}) \ \& \ K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}) \rightarrow H\left(u^{(n)}, u^{(\eta(n))}\right)].$$

并且，

$$\mathfrak{A}_\eta \ \& \ F^{(M)} \rightarrow \left[F\left(u^{(n-1)}, u^{(n)}\right) \& \ H\left(u^{(n-1)}, u^{(\eta(n-1))}\right) \ \& K\left(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}\right) \rightarrow H\left(u^{(n)}, u^{(\eta(n))}\right)\right].$$

在第一个蕴涵符的右侧是$\mathfrak{A}_\eta$公理的倒数第二个表达式，用$u^{(n)}$、 $u^{(n-1)}$、$u^{(\eta(n))}$和$u^{(\eta(n-1))}$的值分别替换w、v、z和x。将这两个公式合并

Hence $\mathfrak{A}_\eta \,\&\, F^{(M)} \rightarrow H(u^{(n)}, u^{(\eta(n))}).$

因此，$\mathfrak{A}_\eta \,\&\, F^{(M)} \rightarrow H\left(u^{(n)}, u^{(\eta(n))}\right)$。。

这就完成了证明的归纳部分。图灵还未证明公式在0处为真，下面就开始证明。

Also $\mathfrak{A}_\eta \,\&\, F^{(r)} \rightarrow H(u, u^{(\eta(0))}).$

同样，$\mathfrak{A}_\eta \,\&\, F^{(r)} \rightarrow H\left(u, u^{(\eta(0))}\right)$。。

这仅仅是将公理的项$\eta(0)$替换成了r。现在我们知道，对于任何n公式为真。

Hence for each n some formula of the form

$$\mathfrak{A}_\eta \,\&\, F^{(M)} \rightarrow H(u^{(n)}, u^{(\eta(n))})$$

is provable.

因此，对于每一个n，符合下列形式的公式

$$\mathfrak{A}_\eta \,\&\, F^{(M)} \rightarrow H\left(u^{(n)}, u^{(\eta(n))}\right)$$

是可证明的。

现在有必要证明，对于（不是图灵在下一句话中说的$\eta(u)$），公理蕴涵着H的否定。

Also, if $M' \geq m$ and $M' \geq m$ and $m \geq \eta(u)$, then

$$\mathfrak{A}_\eta \ \& \ F^{(M')} \rightarrow G(u^{\eta((n))}, u^{(m)}) \vee G(u^{(m)}, u^{(\eta(n))})$$

同样，如果 M'，那么

$$\mathfrak{A}_\eta \ \& \ F^{(M')} \rightarrow G\left(u^{(\eta(n))}, u^{(m)}\right) \vee G\left(u^{(m)}, u^{(\eta(n))}\right)$$

也就是说，如果 $m \neq \eta(n)$，那么m或者大于或者小于$\eta(n)$。

[258]
and

$$\mathfrak{A}_\eta \ \& \ F^{(M')} \rightarrow \left[\{G(u^{(\eta(n))}, u^{(m)}) \vee G(u^{(m)}, u^{(\eta(n))}) \ \& \ H(u^{(n)}, u^{(\eta(n))}\} \rightarrow (-H(u^{(n)}, u^{(m)}))\right].$$

并且

$$\mathfrak{A}_\eta \ \& \ F^{(M')} \rightarrow \left[\left\{G\left(u^{(\eta(n))}, u^{(m)}\right) \vee G\left(u^{(m)}, u^{(\eta(n))}\right) \ \& H\left(u^{(n)}, u^{(\eta(n))}\right)\right\} \rightarrow \left(-H\left(u^{(n)}, u^{(m)}\right)\right)\right].$$

第一个蕴涵符的右边是对 $\mathfrak{A}_\eta$公理的最后一个表达式的重新

描述，只是稍做了调整，以$u^{(n)}$, $u^{(m)}$和$u^{(\eta(n))}$分别代替了w、t和z。将这两个公式合并，

Hence $\mathfrak{A}_\eta \; \& \; F^{(M')} \rightarrow \left(-H\left(u^{(n)}, u^{(m)}\right)\right).$

The conditions of our second definition of a computable function are therefore satisfied.

因此 $\mathfrak{A}_\eta \; \& \; F^{(M')} \rightarrow \left(-H(u^{(n)}, u^{(m)})\right).$

对于可计算函数的第二个定义的条件也满足了。

这里的“第二个定义”是指本书第220页以“一种等价的定义是这样的”开始的等价定义。他已经表明，可以创建以下形式的两个命题：

$$\mathfrak{A}_\eta \; \& \; F^{(M)} \rightarrow H\left(u^{(n)}, u^{(\eta(n))}\right)$$

$$\mathfrak{A}_\eta \; \& \; F^{(M')} \rightarrow \left(-H\left(u^{(n)}, u^{(m)}\right)\right)$$

对于每一个n和m，其中一个是可证明的。

Consequently η is a computable function.

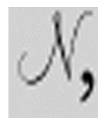
因此，η是一个可计算函数。

下一个证明包含了论文（及本书）中关于图灵机最后出现的那个表。这台机器将用来证明图灵的定理(iii)。该定理声称，如果$ø(m, n)$是一个两个整数变量的可计算函数，那么$ø(n, n)$是n的可计算函数。

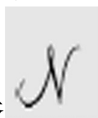
这个证明展示了计算一个定义域为自然数而值域为实数的函数

的技巧。对相继的参数（n等于0、1、2等）开始计算这个函数，而当计算完n个数位后，这个函数的计算就停止了。

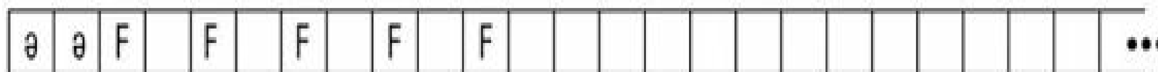
Proof of a modified form of (iii).

Suppose that we are given a machine  which, starting with a tape bearing on it əə followed by a sequence of any number of letters "*F*" on *F*-squares and in the *m*-configuration *b*, will compute a sequence γn depending on the number n of letters "*F*".

对于(iii)变式的证明

假设给定一个机器，该机器初始时的m-格局为b，配备一条以əə开始且在后面的F-格中有任意个字母"F"的纸带，将会根据"F"的个数n来计算序列γ_n。

纸带看起来如下：



这台机器最有趣的地方在于，它是图灵在这篇文章中提到的与克莱尼和戴维斯描述的重构图灵机的工作方式最接近的一次。这台机器读入纸带上以一类字母"F"编码的非负整数，然后计算此整数的函数。克莱尼和戴维斯的机器执行的是数论函数，但在图灵永远运行机器的精髓里，他隐含地假定γ_m序列更加复杂——可能是n的立方根。如果机器以读取5个"F"字符开始（像上面展示的纸带中那样），它将会计算5的立方根。

If $øn(n)$ is the *m*-th figure of γn then the sequence β whose *n*-th figure is $øn(n)$ is computable.

如果$ø_n(m)$是γ_n第m个数字，那么第n个数字为$ø_n$的β序列是可计

算的。

例如，函数$ø_5$返回5的立方根的第12个二进制数字。图灵给出了一个新的可计算序列β，对每一个n包含了$ø_n$的一个数字。在我举的例子中，β序列的第一个数字是1的立方根的第一个数字，第二个数字是2的立方根的第二个数字，以此类推。

图灵一开始已经假设了这台机器（无论是一个立方根机还是其他不同的东西）是存在的。接下来他将改造这台机器，通过改变一些现有的指令并增加一些新的指令，来创造一台稍微不同的机器。

新的机器必须能够以可变数目的F字符运作：一开始没有字符，然后是一个F，两个F字符，等等。在计算了n个（n是字符F的个数加1）数位后，它必须停止计算，然后以一个额外的管理其行为的F字符重新开始。新的机器不会在连续的F-格中打印数字0和1。虽然它们将从左至右按顺序打印，但会淹没在机器的其他输出中。

图灵希望他的原始机器是一个标准形式，这样修改起来就容易了。

We suppose that the table fir $\mathcal{N}$ has been written out in such a way that in each line only one operation appears in the operations column.

同样假设$\mathcal{N}$的格局表写成了如下的形式：每一行的操作列中只有一个操作。

对于这些改动，图灵需要引入一些新的字符，包括希腊大写字母Ξ（读为xi）和Θ（读为theta）。他同时需要替换表中出现的三个字母。

We also suppose that Ξ, Θ, $\overline{0}$, and $\overline{1}$ do not occur in the table, and we replace ə throughout by Θ, 0 by $\overline{0}$, and 1 by $\overline{1}$.

同样假设Ξ、Θ、$\overline{0}$和$\overline{1}$都没有出现在表中，并将ə替换为Θ，0替换为$\overline{0}$，1替换为$\overline{1}$。

当图灵说“将ə替换为Θ”，并不是指在纸带上替换。纸带上仍然需要以两个中央元音哨兵开始。他意指在格局表中，读到ə时就将其改为Θ。

图灵在这里并没有交代，但机器的修改还要求在机器中并没有用到*h*和*k*，且格局$\mathfrak{c}$、$\mathfrak{u}$、$\mathfrak{u}_1$、$\mathfrak{u}_2$、$\mathfrak{u}_3$、$\mathfrak{v}$、$\mathfrak{v}_1$、$\mathfrak{v}_2$、$\mathfrak{v}_3$对新的定义是适用的。

Further substitutions are then made. Any line of form

$$\mathfrak{A} \qquad \alpha \qquad P\overline{0} \qquad \mathfrak{B}$$

we replace by

$$\mathfrak{A} \qquad \alpha \qquad P\overline{0} \qquad \mathfrak{re}(\mathfrak{B}, \mathfrak{u}, h, k)$$

然后，做进一步的替换。对于以下形式的任何一行

$$\mathfrak{A} \qquad \alpha \qquad P\overline{0} \qquad \mathfrak{B}$$

我们将其替换为

$$\mathfrak{A} \qquad \alpha \qquad P\overline{0} \qquad \mathfrak{re}(\mathfrak{B}, \mathfrak{u}, h, k)$$

被替换的行原来打印0，而不是$\bar{0}$。格局$\mathfrak{A}$和$\mathfrak{B}$，以及扫描符α在此处仅仅是占位符。你可能想到$\mathfrak{re}$是一个“替换”函数，这里$\mathfrak{re}$用k替换了纸带上的第一个h，然后转向格局$\mathfrak{B}$。如果它不能找到一个h，那么转向格局$\mathfrak{u}$。

正如第4节的定义，$\mathfrak{re}$函数使用了依赖于中央元音哨兵的$\mathfrak{f}$函数。这个函数不能调整为用来寻找Θ哨兵的函数。

这就是新机器对原始机器打印每一个数位时的操作方式。当新机器以新数目的F字符开始运作时，纸带上h字符的个数会比F字符的个数多一。因此，机器在转入格局u前会打印h次的$\bar{0}$，而不是0。

and any line of the form

$\mathfrak{A}$	α	$P\bar{1}$	$\mathfrak{B}$

by

$\mathfrak{A}$	α	$P\bar{1}$	$\mathfrak{re}(\mathfrak{B}, \mathfrak{v}, h, k)$

并将以下形式的任何一行

$\mathfrak{A}$	α	$P\bar{1}$	$\mathfrak{B}$

替换为

$\mathfrak{A}$	α	$P\bar{1}$	$\mathfrak{re}(\mathfrak{B}, \mathfrak{v}, h, k)$

格局u和 $\mathfrak{v}$ 需要打印没有上划线的真正的0和1字符，然后为下一数目的F字符准备纸带，并重启机器。图灵将揭示格局 $\mathfrak{u}$ 和 $\mathfrak{v}$ 其实是非常相似的。

and we add to the table the following lines:

$\mathfrak{u}$		$\mathfrak{pe}(\mathfrak{u}_1, 0)$
$\mathfrak{u}_1$	$R, Pk, R, P\Theta, R, P\Theta$	$\mathfrak{u}_2$
$\mathfrak{u}_2$		$\mathfrak{re}(\mathfrak{u}_3, \mathfrak{u}_3, k, h)$
$\mathfrak{u}_3$		$\mathfrak{pe}(\mathfrak{u}_2, F)$

并且我们在表中增加以下几行：

$\mathfrak{u}$		$\mathfrak{pe}(\mathfrak{u}_1, 0)$
$\mathfrak{u}_1$	$R, Pk, R, P\Theta, R, P\Theta$	$\mathfrak{u}_2$
$\mathfrak{u}_2$		$\mathfrak{re}(\mathfrak{u}_3, \mathfrak{u}_3, k, h)$
$\mathfrak{u}_3$		$\mathfrak{pe}(\mathfrak{u}_2, F)$

在格局 $\mathfrak{u}$ 打印了一个真正的0后，$\mathfrak{u}_1$打印一个k和两个Θ，对于机器来说这是一个新的哨兵。格局 $\mathfrak{u}_2$和 $\mathfrak{u}_3$用h替换每个k。（在前一轮的调整中，h字符被替换为k。）对于每一个转

换为h的k，在其末尾同样会打印一个F。在图灵的表中，存在一个无限的循环，因为$\mathfrak{u}_2$总是跳转到$\mathfrak{u}_3$，而$\mathfrak{u}_3$又总是跳转到$\mathfrak{u}_2$。替换函数的形式是：

$$\mathfrak{re}(\mathfrak{u}_3, \mathfrak{b}, k, h)$$

如果没有了可以转换为h的字符k，机器就需要从格局$\mathfrak{b}$重新开始，格局$\mathfrak{b}$是原机器的初始格局。格局$\mathfrak{b}$与之非常相似，除了

ə	ə	Ξ	Θ	Θ	…			0	h	Θ	Θ	F	…			0	h	Θ	Θ	F	F			•••

打印的是1而不是0。

and similar lines with $\mathfrak{D}$ for u and 1 for 0 together with the following line

$$\mathfrak{c} \qquad R, P\Xi, R, Ph \qquad \mathfrak{b}.$$

同样用$\mathfrak{D}$替代u，1替代0，并加上下面一行：

$$\mathfrak{c} \qquad R, P\Xi, R, Ph \qquad \mathfrak{b}.$$

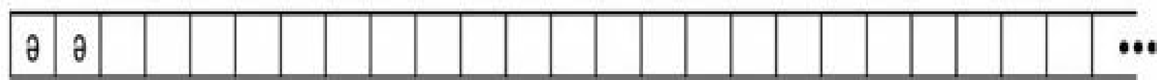
遗憾的是，这里还存在另一个错误。不应该打印h，应该打印新式的哨兵符：

$$\mathfrak{c} \qquad R, P\Xi, R, P\Theta, R, P\Theta$$

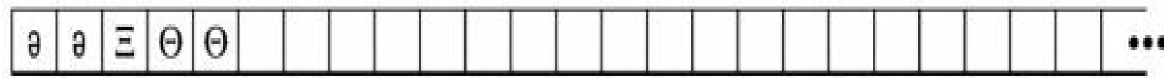
We then have the table for the machine 𝒩′ which computes β. The initial *m*-configuration is 𝔠, and the initial scanned symbol is the second ə.

这样我们就为计算β的机器𝒩′制定了格局表。初始的m-格局是𝔠，初始扫描符是第二个ə。

我们来看看它是否能运作。机器𝒩′是𝒩的修改版，𝒩′主要计算它从纸带上读到的F字符数目的立方根。机器𝒩′以有两个中央元音开头的纸带，从格局𝔠开始：



格局𝔠打印一个Ξ和两个Θ。



Ξ并没有在任何地方用到。现在机器跳到格局𝔟上，两个Θ代替之前的中央元音作为哨兵。这里没有读到F字符，因此计算的是0的立方根。当它需要打印第一个$\overline{0}$时，它也尝试着将第

一个*h*替换为*k*。因为没有*h*字符，所以机器转到格局 。这个格局在纸带上打印一个真正的0，后跟一个*k*和两个Θ：

ə	ə	Ξ	Θ	Θ	…			0	k	Θ	Θ													•••

格局 $_2$和 $_3$组成了一个小循环来将每个*k*替换为*h*，并打印一个F：

ə	ə	Ξ	Θ	Θ	…			0	h	Θ	Θ	F												•••

这里没有可转换为*h*的*k*了，因此机器转向格局 ，并且彻底重新开始计算1的立方根。第一个数字是1，因此机器打印一个

，并且把*h*变为*k*，就这样持续下去。下一个数字是0，因此它

打印一个 。这时没有*h*字符了，因此机器又一次转向格局 打印真正的0及一个*k*和两个Θ。

ə	ə	Ξ	Θ	Θ	…			0	k	Θ	Θ	F	…			0	k	Θ	Θ					•••

现在对于每一个*k*，*k*都转换为了*h*，并且紧接着打印一个F：

ə	ə	Ξ	Θ	Θ	…			0	h	Θ	Θ	F	…			0	h	Θ	Θ	F	F			•••

这里还有一个错误，但我之前并没有修改。F字符需要位于F-格中，因此它们之间需要有空格。不管怎样，已经没有可以转换

为*h*的*k*字符了，因此机器将会转到格局 计算2的立方根。

有了第10节的总结，图灵很满意于他所定义的计算模型，这个模型形式化地包含了所有算法可能需要的东西。现在他已经准备好，要阐述判定性问题无解了。

[1] 本书作者，*Code*：*The Hidden Language of Computer Hardware and Software*（Microsoft Press，1999），第23章。
[2] G. H. 哈代，《纯数学教程》（*A Course of Pure Mathematics*，10th edition，Cambridge University Press，1952，中文版已由人民邮电出版社出版），30。戴德金定理并没有包含在哈代的第1版（1908）中，但出现在第6版（1933）。我并没有仔细检查不同版本的差别。图灵在1931年去剑桥大学之前读了哈代的书，那时他可能得到的是第5版（1928）。哈代的书并不是图灵得知戴德金定理的唯一来源。罗素所著《数学的原理》（*The Principles of Mathematics*，Cambridge University Press，1903）的第34章、*Introduction to Mathematical Philosophy*（George Allen & Unwin，1919，1920）的第7章、E. W. Hobson所著*The Theory of Functions of a Real Variable and the Theory of Fourier's Series*的第1章（图灵在其论文的第8节所引用）都谈到了戴德金定理。1872年，戴德金写的一本小册子"*Stetigkeit und Irrationale Zahlen*"也谈到了这个定理。第一个谈及这个定理的英文译本翻译自戴德金的文集"on the Theory of Numbers"，译者是Wooster W. Beman（Open Court Press，1901，Dover，1963），翻译后的标题改为"Continuity and Irrational Numbers"。这篇译文后来被William Ewald在*From Kant to Hilbert: A Source Book in the Foundations of Mathematics*（Oxford University Press）中重新修订。可参见该书的Vol. II，765-779。
[3] 阿兰·图灵，"On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction"，Proceedings of the London Mathematical Society，2nd Series，Vol. 43（1937），544-546。
[4] 阿隆佐·邱奇，"A Note on the Entscheidungsproblem"，The Journal of Symbolic Logic，Vol. 1，No. 1（ Mar. 1936），40-41。
[5] 阿隆佐·邱奇，"Correction to A Note on the Entscheidungsproblem"，The Journal of Symbolic Logic，Vol. 1，No. 3（Sept. 1936），101-102。

第14章 主要证明

有些数学证明直截了当，有些则需要一些拐弯抹角的方法。后一种当然包含了归谬法，先假设所要证明结论的反论题成立，然后证明依据初始假设推导会导致矛盾。

还有一些方法既不从正面证明，也不从反面证明，它构建出显得有些神秘而放任的详尽结构，看起来似乎完全回避了证明的对象。正当推导的整个中心变得完全模糊，你准备放弃看到结果的希望时，突然灵光浮现，聪明的数学家就完成了证明。

在某种意义上，图灵论文的最后一节最重要，因为图灵正是在这里阐述了“判定性问题不可解”。这在当时是一个非常重要的结论，图灵构建的结构支撑了这一结论（现在称为图灵机的虚构设备），最终比现在吸引我们注意力的实际证明更显得有趣，也更硕果累累。

图灵在论文的第8节为这个证明做了铺垫。虽然这在当时还显得并不重要，但他还是很仔细地证明了，不能设计一台执行通用有限过程的图灵机来判定另一台图灵机是否打印了数字0。中间两节（9和10）表明了，图灵关于机器可计算性的概念与我们习惯的人的可计算性的概念是等价的。

在第11节，图灵介绍了计算机器的功能是如何使用一阶谓词逻辑语言表示的。然后他以这种逻辑构造了一个公式，当且仅当机器打印了数字0时公式才是可证明的。如果公式是可判定的，也就是说，如果可以判定它是否是可证明的，那么我们可以构造一个通用过程来判定机器是否打印了0，我们已经知道这是不可能的。

[259]

11. *Application to the Entscheidungsproblem.*

The results of §8 have some important applications. In particular, they can be used to show that the Hilbert Entscheidungsproblem can have no solution. For the present I shall confine myself to proving this

particular theorem. For the formulation of this problm I must refer the reader to Hilbert and Ackermann's *Grundzüge der Theoretischen Logik* (Berlin, 1931), chapter 3.

11. 在判定性问题中的应用

§8中得到的结论有一些重要的应用，特别是可以用来证明希尔伯特的判定性问题无解。现在，我就来证明这个定理。至于这个问题的构造，我强烈建议读者阅读希尔伯特和阿克曼所著的《数理逻辑原理》（*Grundzüge der Theoretischen Logik*，Berlin，1931）的第3章。

那本书实际上是1928年出版的，共四章120页，第3章大概位于该书的三分之一处，章名是Der engere Funktionenkalkül，即“受限函数演算”，也就是我们今天熟知的一阶谓词逻辑。作者写道：

Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt…. [Das] Entscheidungsproblem muß als das Hauptproblem der mathematischen Logik bezeichnet werden.[1]

当我们知道一个有限操作的过程可以确定任何给定表达式的正确性和可满足性时，这个判定性问题就是可解的……判定性问题理应看作数理逻辑的主要问题。

这里，希尔伯特和阿克曼使用正确性（validity）和可满足性（satisfiability）来表明判定性问题的语义形式。25年后，威廉·阿克曼在他的小书《判定性问题的可解情况》（North-Holland出版公司，1954）中从语义角度继续研究了判定性问题。

图灵使用了一些稍微不同的词语来描述判定性问题。他用**K**（可能代表Kalkül）来表示希尔伯特的受限函数演算。

I propose, therefore, to show that there can be no general process

for determining whether a given formula $\mathfrak{A}$ of the functional calculus **K** is provable, *i.e.* that there can be no machine which, supplied with any one $\mathfrak{A}$ of these formulae, will eventually say whether $\mathfrak{A}$ is provable.

因此我认为，不存在一个通用过程来判定一个给定的函数演算**K**的公式$\mathfrak{A}$是可证的，也就是说，不存在这样的机器：为它提供这些公式中的任意一个$\mathfrak{A}$，而最终能说明$\mathfrak{A}$是否可证明。

通过使用词“可证明”，而不是“正确性”或“可满足性”，图灵表明他正在从句法的角度接近判定性问题。逻辑的句法方法与公理和推理规则系统相关。一个公式被认为是可证明的（也就是说，该公式是定理），只有当它是公理或者可以使用推理规则从公理中推导出来时才成立。一阶逻辑的语义和句法方法等价于哥德尔在1930年提出的完备性定理。

根据第9节和第10节的讨论，图灵现在可以声明，如果没有机器可以实现通用判定过程，那么也不存在人可以进行的通用判定过程。

在1936年，阅读图灵论文的读者并不精通完备性、不完备性和可判定性之间的细微区别，可能会困惑于哥德尔的不完备性证明（出现在一篇标题中含有unentscheidbare Sätze即“不可判定命题”的论文中）和图灵的证明之间的关系。事实上，图灵在论文的第一页已经说了“可以得到与哥德尔的结论大致相似的结论”（本书第57页）。他需要在这个主题上更详尽地阐述一下。

It should perhaps be remarked that what I shall prove is quite different from the well-known results of Gö[†]. Gödel has shown that (in the formalism of Principia Mathematica) there are propositions $\mathfrak{A}$

such that neither $\mathfrak{A}$ nor — $\mathfrak{A}$ is provable. As a consequence of this, it is shown that no proof of consistency of Principia Mathematica (or of **K**) can be given within that formalism. On the other hand, I shall show that there is no general method which tells whether a given formula $\mathfrak{A}$ is provable in **K**, or, what comes to the same, whether the system consisting of **K** with — $\mathfrak{A}$ adjoined as an extra axiom is consistent.

[†] *Loc. cit.*

这里需要注意的是，我将要证明的结论与著名的哥德尔的结论[†]非常不同。哥德尔已经表明（在数学原理的形式主义中）存在命题$\mathfrak{A}$，使得$\mathfrak{A}$和-$\mathfrak{A}$不可证明。这个结论的结果表明，在该形式范围内不能给出数学原理（或者**K**）的一致性的证明。另一方面，我将展示的是不存在一个通用方法来判定一个给定的公式在**K**中是否可证明，或者换一种表达方式，由**K**和一条额外公理公-$\mathfrak{A}$组成的系统是否一致。

[†] *Loc. cit.*

哥德尔的定理和图灵的定理从相反的方向来处理可判定性。哥德尔的定理表明，存在既不能被证明又不能被反驳的命题。这些命题称为不可判定的。

图灵提及的“一般方法”是一个判定过程：对任何公式进行分析并判定其可证明还是不可证明的算法。图灵将证明不存在这样的通用判定过程。

即使存在不可判定命题，我们也能想象存在一个判定过程。在分析

哥德尔的不可判定命题时，需要意识到命题及其反命题都是不可证明的。

If the negation of what Gödel has shown had been proved, *i.e.* if, for each $\mathfrak{A}$, either $\mathfrak{A}$ or — $\mathfrak{A}$ is provable, the we should have an immediate solution of the Entscheidungsproblem. For we can invent a machine $\mathcal{K}$ which will prove consecutively all provable formulae. Sooner or later $\mathcal{K}$ will reach either $\mathfrak{A}$ or — $\mathfrak{A}$. If it reaches $\mathfrak{A}$, then we know that $\mathfrak{A}$ is provable. If it reaches — $\mathfrak{A}$, then, since **K** is consistent (Hilbert and Ackermann, p. 65), we know that $\mathfrak{A}$ is not provable.

如果哥德尔所宣称命题的反命题已经被证明，即如果对于每一个$\mathfrak{A}$，$\mathfrak{A}$或-$\mathfrak{A}$都可证明，那么我们应该能够立即得到判定性问题的解。因为我们可以发明一台机器$\mathcal{K}$，它可以相继证明所有可证明的公式，这台机器早晚会证明$\mathfrak{A}$或-$\mathfrak{A}$。如果证明了$\mathfrak{A}$，那么我们就知道$\mathfrak{A}$可证明。如果证明了-$\mathfrak{A}$，那么因为**K**是一致的（《希尔伯特和阿克曼》，第65页），所以我们

知道$\mathfrak{A}$是不可证明的。

但是图灵令人捉摸不透的机器$\mathcal{K}$（这是论文最后一次提到）显然不是希尔伯特在将判定性问题形式化时所想的。不管任何假想的判定过程是如何的“机械”或者“像机器”，它仍然被看成是人可管理的，而不是一台需要上千年的处理时间和世界上所有内存资源的计算机。

哥德尔的结论并没有奠定一个通用判定过程的基础，图灵的证明仍然是必须的。

Owing to the alsence of integers in **K** the proofs appear somewhat lengthy. The underlying ideas are quite straightforward.

Corresponding to each computing machine $\mathcal{M}$ we construct a formula Un ($\mathcal{M}$) and we show that, if there is a general method for determining whether Un ($\mathcal{M}$) is provable, then there is a general method for determining whether $\mathcal{M}$ ever prints 0.

由于**K**中不存在整数，因而证明会显得有点冗长，但根本的思想还是相当直接的。

对应于每个计算机器$\mathcal{M}$，我们都构造一个公式Un($\mathcal{M}$)，并且表明，如果存在一个通用的方法可以判定Un($\mathcal{M}$)是否可证

明，则存在一个通用的方法来判定$\mathcal{M}$是否打印了0。

图灵是不是在这里过早地抖了包袱，用“不可判定”（Undecidable）来命名公式Un呢？这个Un($\mathcal{M}$)公式起到了反例的作用——没有可以成功进行分析的通用判定过程。

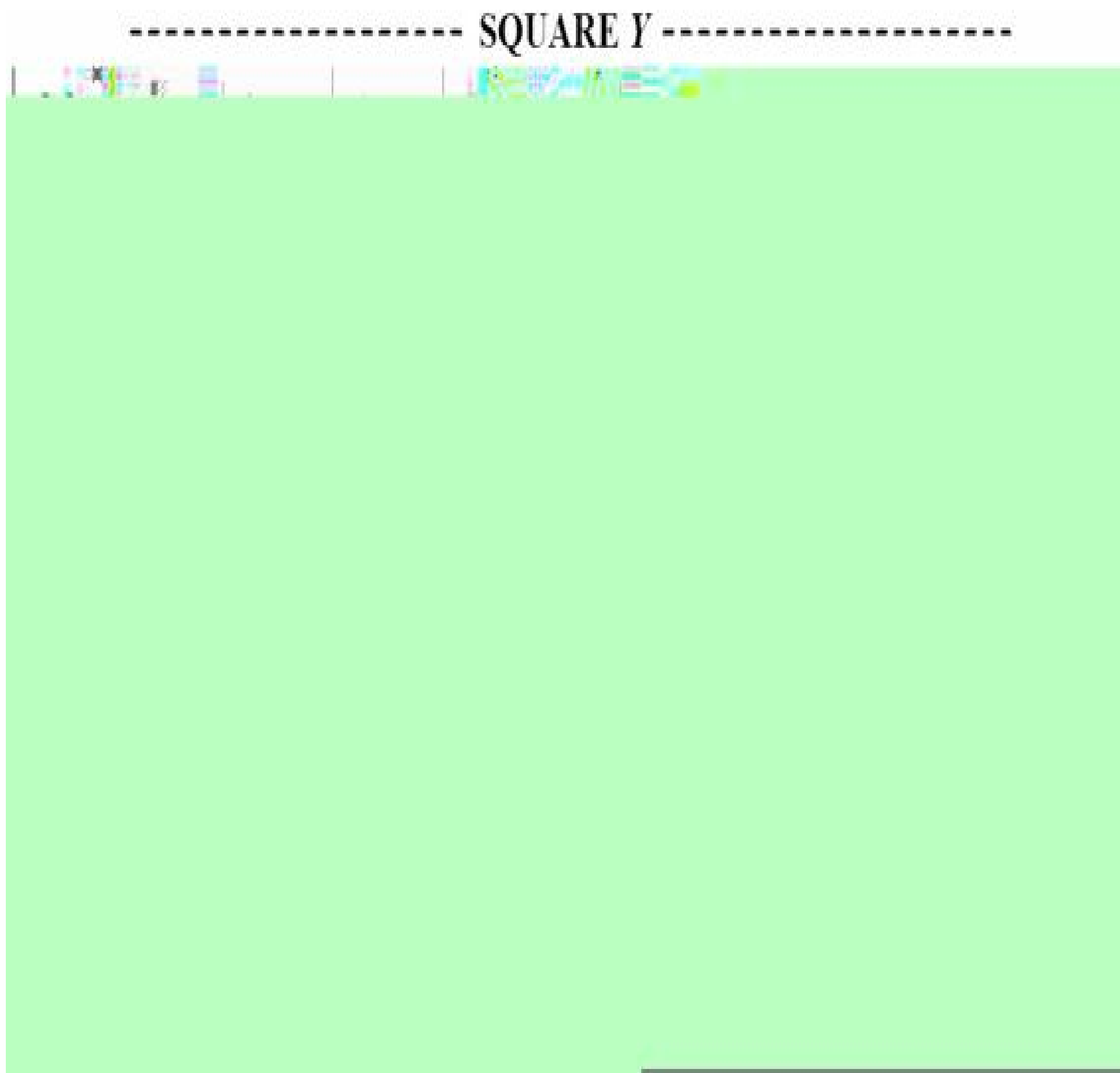
你可能会想起，机器包含了一系列与操作相关的格局。在图灵介绍通用机的最初几页，他使用“指令”一词来表示这些组成机器的基本元素。

图灵需要用一阶逻辑语言来表示这台计算机器。每一个指令将会转换为一个公式，这个公式表明了指令是如何影响机器的完全格局的。你可能还记得，完全格局是在机器进行每一次移动后纸带的快照。完全格局也包括机器的下一个m-格局和下一个扫描符。

图灵首先给出几个命题函数（用现代的术语说就是谓词）。和所有的命题函数一样，它们存在真值，或真或假。在所有情况下，这些函数的参数都是非负整数，用来标注纸带上的方格及完全格局。我们假定，纸带上的方格是以0开始编号的，这表明纸带是有起始的，并且只在一个方向上无限延伸。

完全格局也假设为是依次标号的。数字对(x, y)表示完全格局x中的特定格y。尽管我会在例子中使用实际的数，但在图灵的讨论中没有出现任何数字或例子。

我们回想一下图灵的样机I，它每隔一格交替打印0和1。下面是前10个完全格局。我已经在最左列用数字标识了这些完全格局（标题CCx意思是“完全格局x”），并在顶部用数字标识了纸带上的方格。同时，我用带下标的q代替了图灵原来用于表示格局的字母。



这台机器只是每隔一格打印，因此只有偶数编号的方格才包含数字。出现在方格中的m-格局表明，这个方格会在这个完全格局中被扫描。

图灵的许多命题函数都包含了作为函数名一部分的下标。这些命题函数是被独立定义的，很快你会看到图灵是如何使用它们来描述整台机器的。

The interpretations of the propositional functions involved are as follows:

$R_{S_l}(x,y)$ is to be interpreted as “in the complete configuration x (of

) the symbol on the square *y* is *S*”.

所涉及的命题函数的解释如下：

$R_{S_l}(x, y)$可以解释为“在（的）完全格局*x*中，*y*格中的符号是*S*”。

结尾引号前的*S*漏了下标*l*。回想一下，S_0是一个空格，S_1是0，S_2是1。在样机I中，$R_{S_2}(5, 2)$为真，因为数字1出现在完全格局5的第2格中；$R_{S_2}(5, 6)$为假，因为数字1没有出现在完全格局5的第6格中。

[206]

I(*x*, *y*) is to be interpreted as “in the complete configuration *x* the square *y* is scanned”.

I(*x*, *y*)可以解释为“在完全格局*x*中，*y*格会被扫描“。

在样机I中，函数*I*(6, 6)为真，因为在完全格局6中下一个扫描格是6，但*I*(6, *y*)对于其他任何*y*格都为假。

$K_{q_m}(x)$ is to be interpreted as “in the complete configuration *x* the *m*-configuration is q_m.

$K_{q_m}(x)$可以解释为“在完全格局*x*中的m-格局为q_m。

句子末尾少了半个引号。对于样机I，函数K_{q_2} (5)为真，而K_{q_3}和$K_{q_2}(7)$为假。

$F(x, y)$ is to be interpreted as “y is the immediate successor of x”.

$F(x, y)$可以解释为“y是紧邻x的后继”。

用特殊算术记号表示就是$y = x + 1$。使用图灵早先引入的自然数记号，谓词$F(u''', u'''')$为真，但是经过适当的公理化后，$F(u''', u'''')$应该为假。

到目前为止，图灵仅仅讲述了用来描述运行中机器的完全格局的命题函数。他还未将它们与描述真实机器的格局表中的项对应起来。一台机器的标准形式是每行只包括一条打印指令和一条移动指令。格局表的每一行包含一个m-格局q_i、一个扫描符S_j、一个打印符S_k（可以与S_j相同），读写头的移动方向（向左、向右或根本不移动），以及新的m-格局q_l。

图灵接下来给出了他称为Inst（代表“指令”）的概念。这并不是一个命题函数，而是从刚才给出的命题函数得出的表达式的缩写。每一个指令表达式都会与格局表中的一行关联。这个表达式描述了这些m-格局、符号和读写头移动的特定组合对完全格局的影响。

Inst $\{q_i\, S_j\, S_k\, L\, q_l\}$ is to be an abbreviation for

$$(x, y, x', y')\Big\{\big(R_{S_j}(x, y) \,\&\, I(x, y) \,\&\, K_{q_i}(x) \,\&\, F(x, x') \,\&\, F(y', y)\big)$$
$$\rightarrow \Big(I(x', y') \,\&\, R_{S_k}(x', y) \,\&\, K_{q_l}(x')$$
$$\&\, (z)\Big[F(y', z) \vee \big(R_{S_j}(x, z) \rightarrow R_{S_k}(x', z)\big)\Big]\Big)\Big\}.$$

指令$\{q_i\, S_j\, S_k\, L\, q_l\}$是以下表达式的缩写：

$$(x,y,x',y')\Big\{\big(R_{S_j}(x,y)\,\&\,I(x,y)\,\&\,K_{q_i}(x)\,\&\,F(x,x')\,\&\,F(y',y)\big) \rightarrow \Big(I(x',y')\,\&\,R_{S_k}(x',y)\,\&\,K_{q_l}(x') \,\&\,(z)\Big[F(y',z)\,\mathrm{v}\,\big(R_{S_j}(x,z)\rightarrow R_{S_k}(x',z)\big)\Big]\Big)\Big\}.$$

这是三个可能的指令表达式中的一个，适用于读写头向左移动的情况，指令参数中的*L*表达了这个含义。

首字母(x，y，x'，y')是四个全称量词(x) (y) (x') (y')的缩写。约束变量x和x'是两个连续的完全格局。在第一行的最后你会看到谓词*F*（两个谓词中的第一个），表示x'是x的后继。约束变量y和y'是两个邻接的方格。对于向左移动读写头的指令，y'等于y减去1，即y是y'的后继，正如第一行的第二个谓词*F*所表明的。

第一行中的其他三个谓词给出了当前指令的条件。完全格局为x，扫描格为y，扫描符为S_j，格局为q_i。

第二行以一个蕴涵符开始，作用于表达式余下的所有部分。这些是描述由这条指令得到的完全格局的谓词。在下一个完全格局x'中，将扫描y'格，y格中的当前符号为S_k，新的m-格局为q_l。

指令在第三行以一个表达式结束，该表达式表明除了y格外的所有其他格都必须保持不变。这些格用约束变量z表示。z要么是y'的后继（此时z等于y并将被打印）或者……但是图灵余下的公式是错的。他说在其他所有的格中，S_j替换为S_k，而这没有意义。

对指令表达式更好的阐述是在原论文发表大约一年后，图灵发表在《伦敦数学学会集刊》上的论文修订版。修订版（本书第295页）也不是完全正确的，最后一行应该是：

$$\&(z)[F(y',z)\mathrm{v}([R_{S_0}(x,z)\rightarrow R_{S_0}(x',z)]\&[R_{S_1}(x,z)\rightarrow R_{S_1}(x',z)] \&\cdots\&[R_{S_M}(x,z)\rightarrow R_{S_M}(x',z)])])\}$$

R函数的下标$S_0, S_1,\ldots, S_M$都是$\mathcal{M}$可以打印的符号。也就是说，对于所有的z格，要么z是y'后继（也是说，z就是y——调整过的方格），要么在从完全格局x转到完全格局x'时方格中的符号保持不变。

图灵刚刚展示的指令表达式针对读写头向左移动的情况。

Inst $\{q_i\, S_j\, S_k\, R\, q_l\}$ and Inst $\{q_i\, S_j\, S_k\, N\, q_l\}$

are to be abbreviations for other similarly constructed expressions.

指令$\{q_i\, S_j\, S_k\, R\, q_l\}$和$\{q_i\, S_j\, S_k\, N\, q_l\}$是其他相似构造表达式的简写。

为了多展示点（也为了更方便地参考），下面的框中展示了读写头向右移动时指令的正确表达式。

指令$\{q_i\, S_j\, S_k\, Rq_l\}$是以下表达式的缩写：

$$
\begin{aligned}
(x, y, x', y') \Big\{ & (R_{S_j}(x, y) \,\&\, I(x, y) \,\&\, K_{q_i}(x) \,\&\, F(x, x') \,\&\, F(y, y')) \\
& \rightarrow (I(x', y') \,\&\, R_{S_k}(x', y) \,\&\, K_{q_l}(x') \\
& \qquad \& \,(z)[F(z, y') \vee ([R_{S_0}(x, z) \rightarrow R_{S_0}(x', z)] \,\& \\
& \qquad\qquad [R_{S_1}(x, z) \rightarrow R_{S_1}(x', z)] \,\& \\
& \qquad\qquad \ldots \,\& \\
& \qquad\qquad [R_{S_m}(x, z) \rightarrow R_{S_m}(x', z)])]) \Big\}
\end{aligned}
$$

这个表达式几乎和读写头向左移动的一样，除了$F(y', y)$变为了$F(y, y')$，表明新的扫描格是在上次扫描格的右侧；同样，表达式后面部分的

$F(y', z)$变为了$F(z, y')$，用来表示z等于y。

对于样机I，R_{S_m}函数的下标m的最大值为2，因为机器只打印符号S_0（空）、S_1（0）和S_2（1）。

Let us put the description of $\mathcal{M}$ into the first standard form of §6. This description consists of a number of expressions such as "$q_i\ S_j\ S_k\ L\ q_l$" (or with R or N substituted for L).

让我们把$\mathcal{M}$的描述变成§6中的第一个标准形式。这个描述包含了很多表达式，例如{$q_i\ S_j\ S_k\ L\ q_l$}（或者用R或N代替L）。

实际上，图灵是在第5节设计这个标准形式的。在其论文第241页（本书第126页）的样机I是这样的：

$$q_1 S_0 S_1 R q_2; q_2 S_0 S_0 R q_3; q_3 S_0 S_2 R q_4; q_4 S_0 S_0 R q_1;$$

四个五元组中的每一个为一个指令表达式。

Let us form all the corresponding expressions such as Inst {$q_i S_j S_k L q_l$} and take their logical sum. This we call Des ($\mathcal{M}$).

我们来构造出所有对应的表达式，例如指令{$q_i\ S_j\ S_k\ L\ q_l$}，并计算它们的逻辑和。我们称之为Des($\mathcal{M}$)。

这是机器$\mathcal{M}$的描述。术语“逻辑和”有些模糊，因此在修订版（本书第297页）中图灵使用“合取”代替了这个词。换句话说，所有指令术语都可以用&连接，以一阶逻辑语言表示整个机器。

样机I的Des($\mathcal{M}$)是下列表达式的缩写：

$$\text{Inst}\{q_1S_0S_1Rq_2\}\ \&\ \text{Inst}\{q_2S_0S_1Rq_3\}\ \&\ \text{Inst}\{q_3S_0S_2Rq_4\}\ \&\ \text{Inst}\{q_4S_0S_0Rq_1\}$$

现在，图灵将把Des($\mathcal{M}$)公式代入更大的公式Un($\mathcal{M}$)中，即不可判定公式。这个公式是形如下式的蕴涵关系：

某个机器 → 打印0

公式同时使用了后继函数*F*及命题函数*N*，其中如果参数为自然数，*N*为真。

The formula Un($\mathcal{M}$) is to be

$$(\exists u)\Big[N(u)\ \&\ (x)\big(N(x)\rightarrow(\exists x')F(x,x')\big)\ \&\ (y,z)\big(F(y,z)\rightarrow N(y)\ \&\ N(z)\big)\ \&\ (y)R_{S_0}(u,y)\ \&\ I(u,u)\ \&\ K_{q_1}(u)\ \&\ \text{Des}(\mathcal{M})\Big]\rightarrow(\exists s)(\exists t)\big[N(s)\ \&\ N(t)\ \&\ R_{S_1}(s,t)\big].$$

公式Un($\mathcal{M}$)为：

$$(\exists u)\Big[N(u)\,\&\,(x)\big(N(x)\rightarrow(\exists x')F(x,x')\big)$$
$$\&\,(y,z)\big(F(y,z)\rightarrow N(y)\,\&\,N(z)\big)\,\&\,(y)R_{S_0}(u,y)$$
$$\&\,I(u,u)\,\&\,K_{q_1}(u)\,\&\,\mathrm{Des}(\mathcal{M})\Big]$$
$$\rightarrow(\exists s)(\exists t)\big[N(s)\,\&\,N(t)\,\&\,R_{S_1}(s,t)\big].$$

第四行开始的蕴涵符将公式分为两个部分。每个部分的表达式都用方括号括了起来，方括号前有一个或两个存在量词。

最后一行是最容易的部分：它只是简单地声称存在两个数*s*和*t*，使得字符S_1（0）出现在纸带*t*格上的完全格局*s*中。图灵在第8节证明了，没有任何算法可以告诉我们一台机器是否打印过0，所以我相信你将要看到图灵从天而降的灵感了。

公式Un($\mathcal{M}$)一开始断言存在数*u*（数字0），它既表示第一个完全格局的数字，也表示第一个扫描格。第一行的其余部分仅仅表明对于每一个*x*，都有一个*x*的后继*x′*。

第二行有点长，但仅简单断言了完全格局*u*（0）和纸带上的每个格*y*，*y*格中符号为S_0或空，这是纸带的初始条件。第三行包含一个*I*函数，用来说明在完全格局0中扫描格为0；还有一个*K*函数，用来将初始m-格局设置为q_1。紧跟其后的是描述机器本身的Des($\mathcal{M}$)表达式。

从前面的公式中，图灵抽取出方括号中的部分内容，并给出另一个缩写。

[*N*(*u*)&…& Des($\mathcal{M}$)] many be abbreviated to *A*($\mathcal{M}$).

$[N(u) \& \ldots \& \mathrm{Des}(\mathcal{M})]$可以缩写为$A(\mathcal{M})$。

命题$A(\mathcal{M})$包括了机器的启动条件及对机器的描述，还有一个自由变量u。

在发表的论文修订版中，图灵注意到了隐含在这一证明下的问题，我在第12章也讨论了这个问题。他并没有证明后继是唯一的。因此，他定义了命题函数$G(x, y)$，它在y大于x时为真，以及表达式$\mathcal{Q}$，用来代替皮亚诺公理的P。

$\mathcal{Q}$是下列表达式的缩写：

$$(x)(\exists w)(y, z)\Big\{F(x, w) \,\&\, \big(F(x, y) \rightarrow G(x, y)\big) \\ \&\, \big(F(x, z) \,\&\, G(z, y) \rightarrow G(x, y)\big) \\ \&\, \Big[G(z, x) \vee \big(G(x, y) \,\&\, F(y, z)\big) \vee \\ \big(F(x, y) \,\&\, F(z, y)\big) \rightarrow \big(-F(x, z)\big)\Big]\Big\}$$

这个定义中采用了自然数，这使得简写$A(\mathcal{M})$变得更加简单。这是$\mathcal{Q}$、机器启动条件和机器描述的合取。

$A(\mathcal{M})$是下式的缩写：

$$Q \ \& \ (y)R_{S_0}(u,y) \ \& \ I(u,u) \ \& \ K_{q_1}(u) \ \& \ \mathrm{Des}(\mathcal{M})$$

在$A(\mathcal{M})$中，u是个自由变量，这个变量在Un($\mathcal{M}$)公式中就有约束了：

Un($\mathcal{M}$)是下式的缩写：

$$(\exists u)A(M) \ \rightarrow \ (\exists s)(\exists t)R_{S_1}(s,t)$$

我去除了谓词$N(x)$，因为所有这些命题函数的定义域都隐含假定为自然数。根据谓词R、I、K和F的定义以及公式Inst和Des，图灵说：

When we substitute the meanings suggested on p. 259-60 we find that Un ($\mathcal{M}$) has the interpretation “in some complete configuration of $\mathcal{M}$, S_1 (*i.e.* 0) appears on the tape”.

当我们采用第259～260页暗示的含义时，会发现Un($\mathcal{M}$)可以解释为“在$\mathcal{M}$的某个完全格局中，S_1（也就是0）出现在纸带上”。

因为在Un($\mathcal{M}$)中，蕴涵符左侧的表达式包括了对机器的描述和启动条件，我们假设它为真，它们就是公理。蕴涵符右侧是一个表达式，当机器在运行期间打印了0时为真。因此，如果公式Un($\mathcal{M}$)右侧为真，也就是说机器打印了0，那么其本身也为真；如果机器从未打印0则为假。是否存在一个算法可以确定Un($\mathcal{M}$)是否可证明？如果存在，那么同样也存在一个算法告诉我们任意一台机器是否打印了0。

注意，图灵使用了“暗示的”命题函数的含义。接下来的大部分证明将基于对公式的纯粹句法解释，而不需要我们考虑这些函数的准确含义。

现在，图灵想证明Un($\mathcal{M}$)是可证明的，当且仅当S_1出现在纸带上。他将证明分为两部分，每一部分使用两个引理（辅助的证明），他称之为引理1和引理2，但一开始引理以(a)和(b)命名。

Corresponding to this I prove that

(*a*) If S_1 appears on the tape in some complete configuration of $\mathcal{M}$, then Un($\mathcal{M}$) is provable.

(*b*) If Un($\mathcal{M}$) is provable, then S_1 appears on the tape in some complete configuration of $\mathcal{M}$.

When this has been done, the remainder of the theorem is trivial.

与之对应，我将证明

(a) 如果S_1出现在$\mathcal{M}$的某个完全格局下的纸带上，那么Un($\mathcal{M}$)是可证明的。

(b) 如果Un($\mathcal{M}$)是可证明的，那么S_1出现在$\mathcal{M}$的某个完全格局下的纸带上。

完成这些证明之后，定理的其余部分就很简单了。

比较难的是引理1，图灵又逐字重复了一遍。

[261]

LEMMA 1. *If S_1 appears on the tape in some complete configuration of $\mathcal{M}$, then Un($\mathcal{M}$) is provable.*

We have to show how to prove Un($\mathcal{M}$).

引理1 如果S_1出现在$\mathcal{M}$的某个完全格局下的纸带上，那么Un($\mathcal{M}$)是可证明的。

我们必须说明如何证明Un($\mathcal{M}$)。

和之前一样，u是我们通常所知的0，u'是1，u''是2，$u^{(n)}$的意思是u的右上角有n个撇，表示数字n。

另外，图灵在论文的下一句中引入了一些新的记号，包括3个新函数$r(n, m)$、$i(n)$和$k(n)$。这些函数不是命题函数，因为它们返回整数。参

数*n*是一个完全格局，参数*m*代表纸带上的方格。

函数$r(n, m)$返回在完全格局*n*中出现在*m*格上的字符的索引。索引为0代表空格，1代表0，2代表1，等等，因此$S_{r(n, m)}$代表在完全格局*n*中出现在*m*格上的字符。图灵将把这个以*r*函数为下标的*S*与命题函数*R*相结合：

$$R_{S_{r(n,m)}}(n,m)$$

这个谓词永远为真，图灵将使用含上标的*u*来表示*R*的参数：

$$R_{S_{r(n,m)}}(u^{(n)},u^{(m)})$$

图灵允许在*r*函数中使用*n*和*m*，但是*R*谓词中必须包含$u^{(n)}$和$u^{(m)}$。

图灵引入的第二个新函数是$i(n)$，它返回在完全格局*n*下扫描格的数目，因此谓词：

$$I(u^{(n)},u^{(i(n))})$$

总为真，因为它指的是完全格局*n*和扫描格$i(n)$。第三个新函数$k(n)$返回完全格局*n*中m-格局的索引，因此$q_{k(n)}$就是完全格局*n*中的m-格局。谓词：

$$K_{q_{k(n)}}(u^{(n)})$$

总为真，因为它表明在完全格局*n*中，m-格局为$q_{k(n)}$。

> Let us suppose that in the *n*-th complete configuration the sequence of symbols on the tape is $S_{r(n, 0)}, S_{r(n, 1)},\ldots,S_{r(n, n)}$, followed by nothing but blanks, and that the scanned symbol is the $i(n)$-th, that the *m*-configuration is $q_{k(n)}$.

我们假设在第*n*个完全格局中，纸带上的符号序列为$S_{r(n,0)}$，$S_{r(n,1)}$，… $S_{r(n,n)}$，后面都是空格，并且扫描符是第$i(n)$个，m-格局为$q_{k(n)}$。

在完全格局0中，纸带是完全空白的。完全格局1中，可能有一个非空的符号出现在纸带上。总体而言，对于完全格局*n*，最多有*n*个符号出

现在纸带上（很可能更少）。图灵将这个从第0格开始的符号序列表示为$S_{r(n,0)}, S_{r(n,1)},\ldots S_{r(n,n)}$。如果他实际列出的是$n$+1个字符而不是$n$个，这也没有问题，因为某些$r$函数无疑会返回0，从而指示的是空格。

Then we may form the proposition

$$R_{S_{r(n,0)}}(u^{(n)},u)\ \&\ R_{S_{r(n,1)}}(u^{(n)},u')\ \&\ \ldots \&\ R_{S_{r(n,n)}}(u^{(n)},u^{(n)})$$
$$\&\ I\left(u^{(n)},u^{(i(n))}\right)\ \&\ K_{q_{k(n)}}(u^{(n)})$$
$$\&\ (y)F\left((y,u')\vee F(u,y)\vee F(u',y)\vee\ \ldots\ \vee\ F(u^{(n-1)},y)\vee R_{S_0}(u^{(n)},y)\right),$$

which we may abbreviate to CC_n.

那么我们可以构造这样的命题：

$$R_{S_{r(n,0)}}(u^{(n)},u)\ \&\ R_{S_{r(n,1)}}(u^{(n)},u')\ \&\ \ldots \&\ R_{S_{r(n,n)}}(u^{(n)},u^{(n)})$$
$$\&\ I\left(u^{(n)},u^{(i(n))}\right)\ \&\ K_{q_{k(n)}}(u^{(n)})$$
$$\&\ (y)F\left((y,u')\vee F(u,y)\vee F(u',y)\vee\ \ldots\ \vee\ F(u^{(n-1)},y)\vee R_{S_0}(u^{(n)},y)\right),$$

我们将其缩写为CC_n。

这其实就是“完全格局n”。第一行包括了对应于前n+1个方格中符号的函数的合取，第二行包含了涉及扫描格$i(n)$和m-格局$q_k\,(n)$的函数。

第三行开始，紧跟在全称量词后面的F应该在大括号里面。正如第一行表明了出现在纸带上编号从0至第n格的符号，最后一行表明了编号大于n的格只包含空格。R谓词出现在最后。全称量词y针对纸带上的所有方格。要么u'是y格的后继（即y为0），要么y格是u的后继（即y是1），或者y是u'的后继（即y为2），等等，一直推导到y是n-1的后继（即y是n）。如果y不是上面几种情况中的一种，那么y格包含的是空格。

下面是修正后的版本。

CC_n是下面表达式的简写：

$$R_{S_{r(n,0)}}(u^{(n)},u) \mathbin{\&} R_{S_{r(n,1)}}(u^{(n)},u') \mathbin{\&} \ldots \mathbin{\&} R_{S_{r(n,n)}}(u^{(n)},u^{(n)})$$
$$\mathbin{\&} I(u^{(n)},u^{(i(n))}) \mathbin{\&} K_{q_{k(n)}}(u^{(n)})$$
$$\mathbin{\&} (y)\Big(F(y,u') \vee F(u,y) \vee F(u',y) \vee \ldots \vee F(u^{(n-1)},y) \vee R_{S_0}(u^{(n)},y)\Big),$$

当n等于0时，第一行及第三行的大部分就会消失。CC_0是下面表达式的缩写：

$$I(u,y) \mathbin{\&} K_{q_1}(u) \mathbin{\&} (y)R_{S_0}(u,y)$$

As before, $F(u, u') \mathbin{\&} F(u', u'') \mathbin{\&} \ldots \mathbin{\&} F(u^{(r-1)}, u^{(r)})$ is abbreviated to $F^{(r)}$.

I shall show that all formulae of the form $A(\mathcal{M}) \mathbin{\&} F^{(n)} \rightarrow CC_n$ (abbrevated to CF_n) are provable.

同前一样， $F(u, u')\mathbin{\&}F(u', u'')\mathbin{\&}\ldots\mathbin{\&}F(u^{(r-1)},u^{(r)})$ 缩写为$F^{(r)}$。

我会说明所有形如$A(\mathcal{M}) \mathbin{\&} F^{(n)} \rightarrow CC_n$（缩写成$CF_n$）的公式都是可证明的。

对于$A(\mathcal{M})$公式，你可能会想起，它结合了机器的启动条件（一

条空白的纸带，m-格局为q_1，编号为0的扫描格）及Des($\mathcal{M}$)表达式，后者是机器的说明。Des($\mathcal{M}$)表达式包括了多个指令表达式，每个指令表达式都说明了指令如何改变了扫描格上的符号，改变了m-格局，改变了将要扫描的方格。

图灵为每个完全格局*n*定义了一个称为CF_n的公式。

CF_n为下面公式的缩写：

$$A(\mathcal{M}) \,\&\, F^{(n)} \rightarrow CC_n$$

下面是CF_n的前几个公式：

$$CF_0\colon A(\mathcal{M}) \rightarrow CC_0$$

$$CF_1\colon A(\mathcal{M}) \,\&\, F(u, u') \rightarrow CC_1$$

$$CF_2\colon A(\mathcal{M}) \,\&\, F(u, u') \,\&\, F(u', u'') \rightarrow CC_2$$

$$CF_3\colon A(\mathcal{M}) \,\&\, F(u, u') \,\&\, F(u', u'') \,\&\, F(u'', u''') \rightarrow CC_3$$

…

The meaning of CF_n is “The *n*-th complete configuration of $\mathcal{M}$ is so and so” where “so and so” stands for the actual *n*-th complete configuration of $\mathcal{M}$. That CF_n should be probable is therefore to be expected.

CF_n的意思是"$\mathcal{M}$的第n个完全格局是这样的"，其中"这样"指的是$\mathcal{M}$实际的第n个完全格局。因此，我们可以期待CF_n是可证明的。

图灵用归纳法来说明CF_n公式是可证明的。他首先证明了CF_0是可证明的，接着说明如果CF_n是可以证明的，则CF_{n+1}也是可证明的。

CF_0 is certainly provable, for in the complete configuration the symbols are all blanks the *m*-configuration is q_1, and the scanned square is *u*, *i.e.* CC_0 is

$$(y)\, R_{S_0}(u,y) \;\&\; I(u,u) \;\&\; K_{q_1}\,(u).$$

CF_0当然是可证明的，因为在完全格局中，所有符号都是空格，m-格局为q_1，扫描格为u，即CC_0是

$$(y)\, R_{S_0}(u,y) \;\&\; I(u,u) \;\&\; K_{q_1}\,(u).$$

这是公式CC_0的一个重新排列后的版本，我之前也展示过。$A(\mathcal{M})$的表达式为：

$$Q \;\&\; (y)R_{S_0}(u,y) \;\&\; I(u,u) \;\&\; K_{q_1}(u) \;\&\; \mathrm{Des}(\mathcal{M})$$

$A(\mathcal{M})$包含了与CC_0完全相同的谓词R、I和K。

$A(\mathcal{M}) \rightarrow CC_0$ is then trivial.

We next show that $CF_n \to CF_{n+1}$ is provable for each *n*.

然后易得$A(\mathcal{M}) \to CC_0$。

下面我们证明对于任何*n*，$CF_n \to CF_{n+1}$是可证明的。

看一下CF_n的表达式，可以发现

$$CF_n \to CF_{n+1}$$

是下面公式的缩写：

$$\left[A(\mathcal{M}) \,\&\, F^{(n)} \to CC_n\right] \to \left[A(\mathcal{M}) \,\&\, F^{(n+1)} \to CC_{n+1}\right]$$

这里是归纳证明比较难的部分，但是如果证明了这个蕴涵关系，我们就可以说$CF_0 \to CF_1$是可证明的，$CF_1 \to CF_2$是可证明的，以此推理可知，所有的CF_n表达式都是可证明的。

There are three cases to consider, according as in the move from the *n*-th to the $(n + 1)$-th configuration the machine moves to left or to right or remains stationary. We suppose that the first case applies, *i.e.* the machine moves to the left. A similar argument applies in the other cases.

在从第*n*个格局转换至第*n*+1个格局时需要考虑三种情形：机器是向左移动、向右移动还是保持静止。我们假设是第一种情形，即机器向左移动。对其他情形，类似的论据也成立。

下句话的前一部分，图灵基于之前定义的函数*r*、*i*和*k*定义了整数*a*、*b*、*c*和*d*。但是，这些定义有些混在一起了。

If $r(n, i(n)) = a$, $r(n + 1, i(n+1)) = c$, $k(i(n)) = b$, and $k(i(n + 1)) + d$,

如果
$r(n, i(n))=a$，$r(n+1, i(n+1))=c$，$k(i(n))=b$并且$k(i(n+1))=d$，

在图灵发表的论文修订版中，他解释了这些定义，这些都跟完全格局n有关：

$a = k(n)$，m-格局的索引

$b = r(n, i(n))$，扫描格$i(n)$中符号的索引

$c = k(n+1)$，下一个m-格局的索引

$d = r(n+1, i(n))$，扫描格$i(n)$中新符号的索引

a和c是q的下标；b和d则是S的下标，这些缩写仅仅是为了简化剩余的表达式和接下来的一些条目。

then Des $\mathcal{M}$ must include Inst $\{q_a\ S_b\ S_d\ L\ q_c\}$ as one of its terms, *i.e.*

Des$\mathcal{M}$) $\to$ Inst$\{q_a\ S_b\ S_d\ L\ q_c\}$.

那么Des($\mathcal{M}$)一定包含指令$\{q_aS_bS_dLq_c\}$作为其中一项，即

Des($\mathcal{M}$) $\to$ Inst$\{q_a\ S_b\ S_d\ L\ q_c\}$

因为Des由所有Inst指令的合取组成，因此如果Des为真，那么所有单独的Inst指令也为真，上述蕴涵关系也就成立。A($\mathcal{M}$)是Des和其他表达式的合取，因而

$$A(\mathcal{M}) \to \text{Des}(\mathcal{M})$$

结合这两个蕴涵关系，我们可以得到：

$$A(\mathcal{M}) \rightarrow \text{Inst}\{q_a S_b S_d L q_c\}$$

表达式$F^{(n+1)}$是F谓词合取的缩写，这些F谓词同样也假设为公理。因为$F^{(n+1)}$为真，所以我们可以把它以合取形式加到公式的两边。

Hence

$$A(\mathcal{M}) \ \& \ F^{(n+1)} \rightarrow \text{Inst}\{q_a S_b S_d L q_c\} \ \& \ F^{(n+1)}.$$

But

$$\text{Inst}\{q_a S_b S_d L q_c\} \ \& \ F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

is provable,

因此

$$A(\mathcal{M}) \ \& \ F^{(n+1)} \rightarrow \text{Inst}\{q_a S_b S_d L q_c\} \ \& \ F^{(n+1)}$$。

而

$$\text{Inst}\{q_a S_b S_d L q_c\} \ \& \ F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

是可证明的,

这里需要修正一下：左侧的合取必须包括 $\mathcal{Q}$ ，用来保证后继的唯一性：

$$\text{Inst}\{q_a S_b S_d L q_c\} \,\&\, \mathcal{Q} \,\&\, F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

如果把Inst表达式中的下标a、b、c和d替换掉，将得到一个特别的Inst指令，它使得完全格局n前进至完全格局(n+1)。

$$\text{Inst}\{q_{k(n)} S_{r(n,i(n))} S_{r(n+1,i(n))} L q_{k(n+1)}\} \,\&\, \mathcal{Q} \,\&\, F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

这个公式等价于

$$(CC_n \,\&\, \text{Inst}\{q_{k(n)} S_{r(n,i(n))} S_{r(n+1,i(n))} L q_{k(n+1)}\} \,\&\, \mathcal{Q} \,\&\, F^{(n+1)}) \rightarrow CC_{n+1}$$

直观上看这很明显：特别的Inst指令使CC_n前进到CC_{n+1}，而CC_{n+1}可以由CC_n与这一指令的合取得到。但是，要通过对相关的命题函数进行操作来说明这是可证明的，将是非常繁琐的，因为Inst和CC_n缩写都非常复杂。

图灵刚刚提出的两个命题是可证明的，

and so therefore is

$$A(\mathscr{M})\ \&\ F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

因此

$$A(\mathscr{M})\ \&\ F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$$

这是X → (Y → Z)形式的命题，很容易说明这与X → (Y → Z)等价，因此：

$$\left(A(\mathscr{M})\ \&\ F^{(n+1)} \rightarrow CC_n\right) \rightarrow \left(A(\mathscr{M})\ \&\ F^{(n+1)} \rightarrow CC_{n+1}\right)$$

回想一下，带上标的F是F谓词合取的缩写，因此$F^{(n+1)} \rightarrow F^{(n)}$且。$\left(A(\mathscr{M})\ \&\ F^{(n+1)}\right) \rightarrow \left(A(\mathscr{M})\ \&\ F^{(n)}\right)$。

[262]

and

$$\left(A(\mathscr{M})\ \&\ F^{(n)} \rightarrow CC_n\right) \rightarrow \left(A(\mathscr{M})\ \&\ F^{(n+1)} \rightarrow CC_{n+1}\right),$$

并且

$$\left(A(\mathscr{M})\ \&\ F^{(n)} \rightarrow CC_n\right) \rightarrow \left(A(\mathscr{M})\ \&\ F^{(n+1)} \rightarrow CC_{n+1}\right),$$

这两个括号中的表达式都能表示成CF_n缩写形式。

i.e.

$$CF_n \rightarrow CF_{n+1}.$$

CF_n is probable for each n.

即$CF_n \rightarrow CF_{n+1}$。

对于每个n，CF_n都是可证明的。

至此，给出了说明CF_n是可证明的归纳证明，但我们还没有完成引理的证明，因为我们真正需要证明的是Un($\mathcal{M}$)。

Now it is the assumption of this lemma that S_1 appears somewhere, in some complete configuration, in the sequence of symbols printed by $\mathcal{M}$; that is, for some integers N, K,

现在根据引理假设，S_1在某些完全格局中出现在$\mathcal{M}$所打印的符号序列的某处，也就是说，对某些整数N、K，

这里N是完全格局的序号，K是纸带上的方格，

CC_N has R_{S_1} $(u^{(N)}, u^{(K)})$ as one of its terms, and therefore $CC_N \rightarrow R_{S_1}$ $(u^{(N)}, u^{(K)}$ is provable.

R_{S_1} $(u(^{N)}, u^{(K)}$ 是 CC_N 中的一项，因此$CC_N \rightarrow R_{S_1}$ $(u^{(N)}, u^{(K)})$是可证明的。

这个成立是因为合取式中任意一项都能推出该项成立。

We have then

$$CC_N \to R_{S_1}(u^{(N)}, u^{(K)})$$

and $A(\mathscr{M})\ \&\ F^{(N)} \to CC^N.$

那么我们可以得到

$$CC_N \to R_{S_1}(u^{(N)}, u^{(K)})$$

并且 $A(\mathscr{M})\ \&\ F^{(N)} \to CC^N$。

*CC*的上标应该是下标，这是CF_N的定义，图灵刚刚证明了CF_N对于所有的*N*都可证明（尽管之前他用的是小写的*n*，而不是大写的*N*）。

到目前为止，图灵一直在处理包含自由变量*u*的公式。

We also have

$$(\exists u)A(\mathscr{M}) \to (\exists u)(\exists u')\dots(\exists u^{(N')})\big(A(\mathscr{M})\ \&\ F^{(N)}\big),$$

where $N' = \max(N, K)$.

我们同样有

$$(\exists u)A(\mathscr{M}) \to (\exists u)(\exists u')\dots(\exists u^{(N')})\big(A(\mathscr{M})\ \&\ F^{(N)}\big),$$

其中 $N' = \max(N，K)$。

实际上，*K*（指纸带上出现0的方格）永远不可能比*N*（完全格局）大，因此*N’*总是等于*N*。右侧的表达式$A(\mathscr{M})\ \&\ F^{(N)}$只是为了推导出$CC_N$，我们刚刚证明了$CC_N$可以推导出 $R_{S_1}(u^{(N)}, u^{(K)})$。

And so

$$(\exists u)A(\mathscr{M}) \to (\exists u)(\exists u')\ldots(\exists u^{(N')})R_{S_1}(u^{(N)}, u^{(K)}),$$

因此

$$(\exists u)A(\mathscr{M}) \to (\exists u)(\exists u')\ldots(\exists u^{(N')})R_{S_1}(u^{(N)}, u^{(K)}),$$

R函数并不要求从u至$u^{(N')}$的所有整数都存在，仅仅需要$u^{(N)}$和$u^{(K)}$存在即可。因此大部分存在量词都可以去除，剩下的就是：

$$(\exists u)A(\mathscr{M}) \to (\exists u^{(N)})(\exists u^{(K)})R_{S_1}(u^{(N)}, u^{(K)}),$$

$$(\exists u)A(\mathscr{M}) \to (\exists u^{(N)})(\exists u^{(K)})R_{S_1}(u^{(N)}, u^{(K)}),$$

如果我们用s代替$u^{(N)}$，t代替$u^{(K)}$，就得到

$$(\exists u)A(\mathscr{M}) \to (\exists s)(\exists t)R_{S_1}(s, t),$$

$$(\exists u)A(\mathscr{M}) \to (\exists s)(\exists t)R_{S_1}(s, t),$$

这恰好就是Un($\mathscr{M}$)的定义，你可以在第254页的框中看到。但是，它并不是图灵在其原始论文中对Un($\mathscr{M}$)的定义。他之前把$N(s)$和$N(t)$写在了蕴涵符右侧的表达式中，但那些谓词仅仅表明s和t是自然数，而我们已经推导出这一事实了。

以“如果S_1出现在$\mathcal{M}$的某个完全格局下的纸带上”作为前提，我们刚刚证明了公式Un($\mathcal{M}$)。

i.e. Un($\mathcal{M}$) is provable.

This completes the proof of Lemma 1.

即Un($\mathcal{M}$)是可证明的。

至此，完成了引理1的证明。

第二个引理的证明更简短，只需要使用之前定义的命题函数来解释这个公式。

LEMMA 2. *If* Un($\mathcal{M}$) *is provable, then S_1 appears on the tape in some complete configuration of* $\mathcal{M}$.

If we substitute any propositional functions for function variables in a provable formula, we obtain a true proposition. In particular, if we substitute the meanings tabulated on pp. 257-260 in Un ($\mathcal{M}$), we obtain a true proposition with the meaning “S_1 appears somewhere on the tape in some complete configuration of $\mathcal{M}$".

引理2 如果Un($\mathcal{M}$)是可证明的，那么S_1出现在M的某个完全格局下的纸带上。

如果我们在可证明的公式中用任意命题函数来代替函数变量，那么就会得到一个真命题。特别地，如果我们采用含义“S_1出现在$\mathcal{M}$的某个完全格局下的纸带上”而非第259~260页的Un($\mathcal{M}$)中的含义，就能得到一个真命题。

现在，图灵已经证明，Un($\mathcal{M}$)可证明当且仅当 S_1出现在机器$\mathcal{M}$的某个完全格局下的纸带上。

We are now in a position to show that the Entscheidungsproblem cannot be solved. Let us suppose the contrary. Then there is a general (mechanical) process for determining whether Un($\mathcal{M}$) is provable. By Lemmas 1 and 2, this implies that there is a process for determining whether $\mathcal{M}$ ever prints 0, and this is impossible, by § 8. Hence the Entscheidungsproblem cannot be solved.

我们现在来证明判定性问题不可解。先假设其否定命题成立，那么就存在一个通用（机械）过程可以判定Un($\mathcal{M}$)是否可证明。根据引理1和引理2，这表明存在一个过程可以确定机器$\mathcal{M}$是否打印了0，但这在§8中已经提及是不可能的。因此判定性问题

不可解。

回想一下，这很简单，你同意吧？

Un($\mathcal{M}$)是个相当复杂的公式，这一点也不奇怪。如果它比较简单，那么它就可能会被判定过程分析。相反，Un($\mathcal{M}$)中包含了*A*($\mathcal{M}$)，*A*($\mathcal{M}$)中包含了*Q*和Des($\mathcal{M}$)，Des($\mathcal{M}$)还是组成机器的所有Inst项的合取。每个Inst项有五个全称量词，*Q*包含三个全称量词和一个存在量词，*A*($\mathcal{M}$)包含另一个全称量词，Un($\mathcal{M}$)含有三个存在量词。

如果这个复杂的量词嵌套中仅包含一元谓词，也就是说，谓词中仅含一个参数，那么它不会影响命题的可解性。1915年，利奥波德·勒文海姆（1878—1957）证明了仅包含一元谓词的命题是可判定的。[2]

到图灵写这篇论文时，人们在为特殊情形的公式寻找判定过程方面已经取得了许多进展。通常，应用判定过程时，公式一般首先转换为前束范式，也就是，通过改变公式让所有（非否定形式）的量词都可以移到公式的开始处，位于一个称为“矩阵”的不含任何量词的表达式之前。

在那些年里，许多数学家已经发现了前束范式形式的很多类公式的判定过程，它们都是以一个特殊形式的量词开头。在有关判定性问题的文献中[3]，都是使用 $\forall$ 和 $\exists$ 符号表示全称量词和存在量词。上标的数字代表了一个特定量词的数目，星号可以是任何数字。

1928年，保罗·贝奈斯和摩西·施隆芬克尔（1889—1942）发表了以 $\exists^*\forall^*$ （任意数量的存在量词，后跟任意数量的全称量词）开头的命题的判定过程。同样在1928年，威廉·阿克曼给出了

$\exists^* \forall \exists^*$的判定过程（任意数量存在量词后跟一个全称量词，再接任意数量的存在量词）。1932年，哥德尔给出了任意数量的存在量词间包含两个全称量词的判定过程：$\exists^* \forall^2 \exists^*$。

人们还探索了与判定性问题相联系的一类问题：化简类。化简类由所有以特定量词模式开头的命题组成，仅当所有命题都存在判定过程，各种化简类中的命题存在判定过程。1920年，斯科莱姆证明$\forall^* \exists^*$定义了一个化简类。1933年，哥德尔将范围缩小到$\forall^3 \exists^*$。

在图灵和邱奇的证明之前，化简类是否存在判定过程是未知的，仅仅知道如果存在一个化简类的判定过程，那么也会存在一个通用判定过程。图灵和邱奇的论文的结果表明，这些化简类是不可判定的。1932年，哥德尔给出了一个以$\exists^* \forall^2 \exists^*$为前缀的命题的判定程序，并将其推广到$\forall^2 \exists^*$。在图灵的证明出现之后，我们可知任何以$\forall^3 \exists^*$为前缀的命题是不可判定的。通过增加一个全称量词，形为$\forall^2 \exists^*$的可判定命题变成了形为$\forall^3 \exists^*$的不可判定命题。

在下面的段落中，图灵使用quantor代表量词。

In view of the large number of particular cases of solutions of the Entscheidungsproblem for formulae with restricted systems of quantors, it

[263]

is interesting to express Un($\mathcal{M}$) in a form in which all quantors are at the beginning. Un($\mathcal{M}$) is, in fact, expressible in the form

$$(u)(\exists x)(w)(\exists u_1)\dots(\exists u_n)\mathfrak{B}, \tag{I}$$

where $\mathfrak{B}$ contains no quantors, and n = 6.

对于带受限量词（quantor）的公式的判定性问题的一大类特殊情形的解，把Un($\mathcal{M}$)表示成所有的量词都出现在开始处，这会比较有趣。事实上，Un($\mathcal{M}$)可以表示成

$$(u)(\exists x)(w)(\exists u_1)\ldots(\exists u_n)\mathfrak{B}, \tag{I}$$

其中$\mathfrak{B}$不包含任何量词，并且n = 6。

图灵已经证明以$\forall\exists\forall\exists^6$（使用约定俗成的定义）为前缀的命题构成了一个化简类。带有此类前缀的命题不存在判定过程。

By unimportant modifications we can obtain a formula, with all essential properties of Un($\mathcal{M}$), which is of form (I) with n = 5.

通过改变一些不重要的地方，我们可以得到一个包含所有Un($\mathcal{M}$)必要性质的公式，形式与(I)相同，但n = 5。

在图灵论文的修订版中，图灵标注出这个数字应该是4，因此化简类被进一步限制到$\forall\exists\forall\exists^4$。

1962年，瑞士逻辑学家朱利叶斯·理查德·步琪（1924—1984）

利用图灵机从另一方面着手研究判定性问题，并且成功地使证明简化了一点。[4] 他的研究表明，$\exists \& \forall\exists\forall$ 形式的命题形成的是化简类。（这类命题是两个项的合取，每一项都将自身的量词置于最前面。）步琪的论文同时为证明 $\forall\exists\forall$ 命题形成化简类奠定了基础，这意味着即使对于以 $\forall\exists\forall$ 为前缀的简单命题都不存在通用判定过程。

数学家们发明解决问题的方法惠及了整个世界，当他们证明了某个问题无解时也能起到同样的效果。就如无法仅用尺和圆规三等分一个角，无法求与已知圆面积相等的正方形，无法证明能从欧几里得的前四个公设推导出第五个。对于n大于2的情况，方程$x^n + y^n = z^n$ 没有整数解，无法在系统中建立算术的一致性，以及一阶逻辑没有通用判定过程。

我们可以不再把时间浪费在一个无法做到事情上。知道什么不可能与知道什么可能同样重要。

---

[1] 希尔伯特和阿克曼，*Grundzüge der Theoretischen Logik*（Springer，1928），73，77。

[2] 利奥波德·勒文海姆，“Über Möglichkeiten im Relativekalkül”，*Mathematische Annalen*，Vol. 76（1915），447-470。被译为“On Possibilities in the Calculus of Relatives”，参见Jean van Heijenoort，ed.，*From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*（Harvard University Press，1967），228-251。

[3] 更为出名的是Egon Börger、Erich Grädel和Yuri Gurevich的*The Classical Decision Problem*（Springer，1997）。这本书是撰写判定问题和其部分解决方法论文时应该参考的好书。

[4] J. Richard Büchi，“Turing-Machines and the Entscheidungsproblem”，*Mathematische Annalen*，Vol. 148，No.3（June 1962），201-213。

第15章
λ　演　算

在1983年或是1984年，阿隆佐·邱奇大概80岁时，他被邀请到斯坦福大学的语言和信息研究中心做报告，并参观了中心的Xerox Dandelion计算机。计算机运行的是LISP语言，这是由约翰·麦卡锡（1927—2011）发明的一种编程语言。邱奇被告知，LISP是基于他50多年前发明的λ演算发展而来的。

邱奇坦言他对计算机一无所知，但他的一个学生了解计算机。[1]当然，那时所有人都已经知道阿兰·图灵是谁了。

λ演算是阿隆佐·邱奇在20世纪30年代早期为证明一阶谓词逻辑没有通用判定过程而发明的。图灵在发表关于可计算数和判定性问题的论文之前就对此有所了解了。为此，他在论文中增加了一个附录，用来说明他的方法与邱奇的方法在本质上是等价的，这一附录就是本章的主题。

如果说对λ演算的理念觉得很熟悉，那是因为它们对编程语言的发展影响深远。相当早的时候，人们就发现λ演算与划分为过程式或命令式的编程语言之间存在着结构关系，例如早期的编程语言ALGOL[2]，基于这个语言发展出了Pascal和C，同样在C基础上发展出了 C++、Java和C#。用过程式语言编写的程序，其结构围绕着将数据传递给过程（也称为子程序或方法）的理念来组织，过程以各种各样的方式对数据进行处理。

λ演算对LISP、APL、Haskell、Scheme和F#等函数式编程语言有着更直接的影响。在函数式语言中，函数的排列方式更像是链条，后一个函数从前一个函数获得输出。函数式语言通常允许像过程式语言操作数据那样来操作函数。尽管函数式语言没有像过程式语言那样受到主流群体的欢迎，但是最近它们也展露了一些复兴的气息。

阿隆佐·邱奇1903年出生于华盛顿，他职业生涯的大部分时间是在普林斯顿大学度过的。他本科就读于普林斯顿，24岁时获得数学博士学位。在作为国家研究院院士工作两年后，他回到普林斯顿教学，从1929年一直到1967年退休。邱奇后来又在UCLA工作了一段时间，直到1990年。

邱奇是一个勤奋且一丝不苟的人，说话严谨，经常工作到深夜。他的课程经常开始于精心设计的仪式般的擦黑板，有时候还会借助一桶水。在处理数学问题时，他通常会用不同颜色的墨水。当需要更多颜色的时候，他就会按照不同比例混合那些标准颜色的墨水；当完成了一页并要保存起来时，他会用Duco涂满表面。Duco是一种漆，邱奇发现它非常适合保护纸张，因为它不会把纸弄皱。[3]

邱奇指导过31篇博士论文，论文来自于斯蒂芬·克莱尼（1931）、约翰·巴克利·罗瑟（1934）、莱昂·亨金（1947）、马丁·戴维斯（1950）、哈特利·罗杰斯（1952）、雷蒙德·斯穆里安（1959）以及阿兰·图灵（1938）。[4]

通常认为，邱奇建立了符号逻辑协会，因为他是《符号逻辑杂志》的第一位编辑。实际上，他并没有建立这个组织，而是指导杂志取得了辉煌的成功，最突出的是他编纂了重要的逻辑文献书目。

邱奇是在1927年至1929年作为国家研究院院士时开始λ演算的工作的。在那时，数学家们希望理解有效可计算性这个模糊的概念。为了知道数值计算的局限及能力，有必要用形式系统化的方式来定义函数，也就是说用符号、字符串和确定性规则来定义。最好的方法是什么呢？这种方法能否表明这些函数能够充分概括有效可计算性？

邱奇就这一方面的第一篇论文在1931年10月5日被《数学年鉴》收录，并在下一年的4月发表。[5] 在这篇文章中，邱奇用了一个小写的lambda（λ）来表示函数。

邱奇引入新记号的一个动机来自于传统函数表示的特定二义性。看看下面的表达式：

$$x^2+5x+7$$

就表达式本身而言，它在语法上是正确的，但我们仍不知道该对这个表达式做些什么，下面这个就清晰点了：

$$f(x)=x^2+5x+7$$

这是传统的函数标记法，其中x是约束变量（或自变量）。我们可以按照自己的意愿任意改变约束变量，只要与函数表达式中的其他部分不冲突就行：

$$f(y)=y^2+5y+7$$

我们可以使用表达式$f(4)$来表示函数的一个值，用4代替自变量x，并计算函数的值

$$f(4)=(4)^2+5\cdot(4)+7=43$$

当你意识到这一点时可能会吃惊，那就是没有标准的形式可以将函数表达式（也就是y^2+5y+7）与y的具体值一起表示出来。一旦我们用4替换了y，就丢失了自变量信息。如果你需要修复这样的缺点，并发展

出一套表示取某个值的函数的方式，可能会想到这样的东西：

$$[y^2+5y+7](4)$$

这其实不算太差，但是如果表达式中含有多个自变量呢？下面的表示就会让人不明就理：

$$[y^2+5y+18x-2xy+7](4)$$

即使你写成：

$$[y^2+5y+18x-2xy+7](4,5)$$

你假设对x和y按照特定顺序赋值。

在《数学原理》中，怀特海和罗素使用了抑扬符号来表示符合特定函数的类：λy。邱奇想把这个抑扬符号移到变量的前面，像$^\wedge y$一样，但因为书写的原因，这个符号后来改成了lambda[6]：λy。

多年来，邱奇的表示法已经有了发展。在下面的讨论中，我使用图灵在其论文附录中使用的表示法。一元变量的函数用下面这个通用形式表示：

$$\lambda x[M]$$

其中M是包含自变量x的表达式。对于之前的例子，我们可以表示为：

$$\lambda x[x^2+5x+7]$$

一个包含特定自变量值的函数可以写成这个通用形式：

$$\{F\}(A)$$

F是函数，如果F含有一个自变量，那么公式代表了函数，其中A代表函数中的自变量。例如，如果函数中的自变量为x，则通用形式为：

$$\{\lambda x[M]\}(A)$$

将M替换为之前的函数，就变成了：

$$\{\lambda x[x^2+5x+7]\}(A)$$

若x的值为4，则可以将其写为：

$$\{\lambda x[x^2+5x+7]\}(4)$$

现在，我们已经成功地将函数和某个自变量的值表示在一起了。

含有两个自变量的函数有如下一般形式：

$$\{\{F\}(A)\}(B)$$

为了方便且可读性强，它可以简写为：

$$\{F\}(A,B)$$

如果将F替换为一个真正的函数，它看起来会是：

$$\{\lambda x\lambda y[y^2+5y+18x-2xy+7]\}(A,B)$$

我们现在知道A将替代x，B将替代y，它们以前面λ的排列顺序为准。也可以再简写。如果没有歧义，可以去掉花括号，那么

$$\{F\}(A,B)$$

变成

$$F(A,B)$$

这看起来很像常规的函数表示法，但F表达式实际上包含了一些λ：

$$\lambda x\lambda y[M](A,B)$$

邱奇还允许将方括号替换为一个点，紧跟在λ串的后面：

$$\lambda x\lambda y.M(A,B)$$

这个形式会在接下来的大多数lambda表达式中出现。

邱奇在建立了基本的lambda表示法后，引入了常用逻辑算子的表达式和用于转换为等价公式的替换规则。邱奇以形式化方式定义了这些变换的规则，也可归结为：

I。如果新的自变量不与公式中的其他部分冲突，可以改变某个自变量（例如，x换为y）；

II。在公式$\{\lambda x.M\}(N)$中，如果N中不包含x的任何东西，那么可以将N替换到M中任何x出现的位置，这样公式就变成了用N替换原始的x所形成的M；

III。Ⅱ的相反情况仍成立。

在第一篇lambda函数的论文发表一年半后，邱奇发表了第二篇论文[7]。他修正了其给出的假设，并强调：“整个系统的形式字符，使得从符号的本身意义进行抽象，以及将（形式逻辑的）定理证明看做是可以根据一套特定规则在纸上进行推导的游戏成为可能。”[8]这个理念很大程度上符合形式主义的传统。

邱奇也引入了缩写conv来表示“通过转换”，指出利用规则Ⅰ、Ⅱ、Ⅲ可以将一个公式转换为等价形式，例如

$$\lambda x[x^2+5x+7](A)\mathrm{conv}A^2+5A+7$$

邱奇第二篇论文的最后是对正整数的研究。他使用了lambda表示法来定义符号1、后继、加运算、乘运算以及5个皮亚诺公理，并且声明道：“我们的纲领是要在之前所描述内容的基础上建立一套关于正整数的理论，然后，通过已知的方法或是进行适当的改变，将该理论推广到有理数以及实数领域。”[9]

邱奇的学生斯蒂芬·科尔·克莱尼在约翰·巴克利·罗瑟（1907—1989）的帮助下继续下一步的研究。1934年，克莱尼所做的基础工作体现在”形式逻辑的例证”[10]一文中，并简化了多重lambda表示法。之前使用的

$$\lambda x\lambda yM$$

现在可以使用

$$\lambda xyM$$

来表示。

克莱尼于1935年改写并发表了他的博士论文“形式逻辑的正整数理论”，[11]该论文分为两部分。阅读这篇论文之前要先看邱奇的两篇论文以及克莱尼之前的论文，其第二部分也提及了即将发表的邱奇和罗瑟合作的论文。[12]

尽管由邱奇、克莱尼、罗瑟一起发明的λ演算相当全面，既涉及逻辑又涉及算术，但是这里我只想集中探讨一些基本的算术，这样你就可以看到，如何通过单纯的字符运算来实现加法和乘法了。

定义自然数时，我们总是需要从0或1开始。邱奇和克莱尼是以1开始的，下面就是其符号：[13]

$$1 \rightarrow \lambda fx.f(x)$$

这里箭头的意思是“代表”或“全称是”。这个公式看起来似乎有点怪，实际上可能看起来是非常怪，但这仅仅是一个定义，所以现在还没必要追究其意义。更冗长的版本是

$$1 \rightarrow \{\lambda fx[f(x)]\}$$

因此，1实际上是一个含有两个自变量f和x的函数，非常唐突，定义一个数字居然需要额外多出的两个变量。

下面的是后继函数：

$$S \rightarrow \lambda \rho fx.f(\rho(f,x))$$

还是很奇怪吧，我也这么认为。尽管我们希望后继函数中包括自变量，但不希望其中包含3个自变量。

幸运的是，符号2可以定义成你期望的形式：

$$2 \rightarrow S(1)$$

如果我们事实上想对1应用后继函数，就必须保证所有的自变量都是唯一的，因此要使用下面1的等价表达式：

$$1 \rightarrow \lambda ab.a(b)$$

当使用λ表达式时，函数和变量经常互换角色。在下面变换后的公式的推导过程中，我选择性地使用花括号来标记将要在那一步被替换的包含一个自变量的函数。

函数$S(1)$也可以写为$\{S\}1$或者：

$$\{\lambda \rho fx.f(\rho(f,x))\}(\lambda ab.a(b))$$

后继函数的第一个自变量为ρ，因此用1的表达式将函数中的ρ替换掉，λ后的ρ就消失了：

$$\lambda fx.f(\lambda ab.a(b)(f,x))$$

现在这个公式包含另一个函数，该函数中含有两个参数：

$$\lambda fx.f(\{\lambda ab.a(b)\}(f,x))$$

用f替换a，x替换b：

$$\lambda fx.f(f(x))$$

这样就完成了。

尽管1最初定义为：

$$1 \rightarrow \lambda fx.f(x)$$

数字2定义为：

$$2 \rightarrow S(1) \operatorname{conv} \lambda fx.f(f(x))$$

比较转换后的2和1的表达式，你会发觉在点的右侧多出一个f和一对括号。现在使用不同的变量$\lambda ab.a(a(b))$来表示2，并尝试确定下一个后继数$\{S\}(2)$:

$$\{\lambda \rho fx.f(\rho(f,x))\}(\lambda ab.a(a(b)))$$

同样，用2来替代ρ：

$$\lambda fx.f(\{\lambda ab.a(a(b))\}(f,x))$$

用f替换a，x替换b：

$$\lambda fx.f(f(f(x)))$$

这是3的λ表达式。我猜你开始明白这个模式了，我们最想从正整数的抽象表达式中得到的是某种后继性。下面这个记号表明了这一后继性：每一个后继整数都会增加对第一个自变量的嵌套。

克莱尼定义“加”运算为：

$$+\rightarrow\lambda\rho\sigma fx.\rho\left(f,\sigma\left(f,x\right)\right)$$

有点怀疑了吧？我们试着将2和3相加。首先我们需要使所有的自变量不同。我将使用$\lambda ab.a(a(b))$代表2，$\lambda cd.c(c(c(d)))$代表3，因此{+}(2,3)就是：

$$\{\lambda\rho\sigma fx.\rho(f,\ \sigma(f,x))\ \}(\lambda ab.a(a(b)),\lambda cd.c(c(c(d))))$$

在+函数中，用2的公式替换ρ，用3的公式替换σ：

$$\lambda fx.\lambda ab.a\left(a\left(b\right)\right)\left(f,\left\{\lambda cd.c\left(c\left(c\left(d\right)\right)\right)\right\}\left(f,x\right)\right)$$

替换后的3是一个函数，其中用f替换c，x替换d：

$$\lambda fx.\left\{\lambda ab.a\left(a\left(b\right)\right)\right\}\left(f,f\left(f\left(f\left(x\right)\right)\right)\right)$$

现在替换后的2是一个用f替代a，用$f(f(f(x)))$替代b的函数：

$$\lambda fx.f\left(f\left(f\left(f\left(f\left(x\right)\right)\right)\right)\right)$$

到这里，我们就完成了运算。答案与$S(S(S(S(1))))$-致，也就是5。

有意思的是，乘法函数比加法函数还要简单：

$$\times\rightarrow\lambda\rho\sigma x.\rho\left(\sigma\left(x\right)\right)$$

我们来计算2和3的乘法，可以将{×}(2,3)写成：

$$\left\{\lambda\rho\sigma x.\rho\left(\sigma\left(x\right)\right)\right\}\left(\lambda ab.a\left(a\left(b\right)\right),\lambda cd.c\left(c\left(c\left(d\right)\right)\right)\right)$$

用2的公式替换ρ，3的公式替换σ：

$$\lambda x.\lambda ab.a\left(a\left(b\right)\right)\left(\left\{\lambda cd.c\left(c\left(c\left(d\right)\right)\right)\right\}\left(x\right)\right)$$

现在3变成一个用x替代c的函数：

$$\lambda x.\left\{\lambda ab.a\left(a\left(b\right)\right)\right\}\left(\lambda d.x\left(x\left(x\left(d\right)\right)\right)\right)$$

现在2变成一个函数，用右侧的表达式替换a：

$$\lambda x.\lambda b.\lambda d.x(x(x(d)))(\{\lambda d.x(x(x(d)))\}(b))$$

在右侧的函数，用b替换d

$$\lambda x.\lambda b.\{\lambda d.x(x(x(d)))\}(x(x(x(b))))$$

最后替换d：

$$\lambda xb.x(x(x(x(x(x(b))))))$$

这就是6，当然也是2与3相乘的结果。

这些数字的函数定义可以让你做一些比较奇怪的事情，例如

{2}(3）

即：

$$\{\lambda ab.a(a(b))\}(\lambda cd.c(c(c(d))))$$

在你完成了所有费力的替换后，最后会得到：

$$\lambda bd.b(b(b(b(b(b(b(b(b(d)))))))))$$

也就是9，毫无意外，这就是3的二次方。这也就是为什么m的n次方定义为：

$$\lambda mn.nm$$

在这个系统中，乘比加看起来要简单，指数形式是其中最简单的。当邱奇、克莱尼和罗瑟使用λ演算做实验时，他们发现λ表示法可以表示任何他们能想到的东西，这是后来称为λ可定义性的性质。“邱奇一直在推断，并最后明确地提出，λ可定义函数是所有的有效可计算函数。”[14]

库尔特·哥德尔在1933年来到高级研究院。1934年春，他在普林斯顿讲授他的不完备性定理及递归函数，递归函数是从基本原始函数发展出来的。[15] 让哥德尔对递归函数感兴趣的是雅克·赫尔布兰德（1908—1931）在1931年写给他的信，这位年轻聪明的法国数学家在攀登阿尔卑斯山时意外身亡了。

尽管如此，哥德尔在那时仍然坚信，λ函数和递归函数都不足以包含所有我们认为的非正式有效可计算性。

1936年，邱奇发表了“初等数论的不可解问题”[16]，在这篇文章中首次出现了“λ可定义”一词。（之前，克莱尼在使用lambda记号表示逻辑和算术运算时仅仅使用了“可定义”或者“形式可定义”。）邱奇引用了自己之前的论文、克莱尼的两篇论文，以及克莱尼即将发表的探讨递归函数和λ可定义函数之间关系的论文。[17]与哥德尔构造了一个不可判定的命题一样，邱奇使用哥德尔的记数法同样构造了一个不可解的问题。

在此基础上，邱奇在《符号逻辑期刊》（他自己也是该期刊的编辑）的首版上发表了两页的“判定性问题的笔记”，他在结论中写道：“判定性问题的通用情况是不可解的。”[18]这篇文章是在1936年4月15日被杂志接收的，比图灵在1936年5月28日提交到伦敦数学学会的文章早了6周。

1936年夏，图灵可能花了大部分时间阅读我之前提到的邱奇和克莱尼的文章，学习λ演算，并验证它如何与自己的计算机器相关联。据显示，图灵的三页附录在8月28日被伦敦数学学会接收。图灵在最后写上了：“美国新泽西州普林斯顿大学研究生院。”他很期待自己的新家。直到9月23日，他才离开英国前往美国，在29日到达纽约。[19]

Added 28 *August*, 1936.

APPENDIX.

Computability and effective calculability

The theorem that all effectively calculable (λ-definable) sequences are computable and its converse are proved below in outline.

1936年8月28日新增

附录

可计算性和有效可计算性

所有的有效可计算（λ可定义）序列都是可计算的，反之也成立。这个定理的证明将在下面简略给出。

在这里，“简略”的意思是证明中会跳过一些步骤。

It is assumed that the terms “well-formed formula”（W.F.F.) and “conversion” as used by Church and Kleene are understood. In the second of these proofs the existence of several formulae is assumed

without proof; these formulae may be constructed straightforwardly with the help of, *e.g.*, the results of Kleene in “A theory of positive integers in formal logic”, *American Journal of Math.*, 57 (1935), 153-173, 219-244.

假设人们已经理解了邱奇和克莱尼所使用的“合式公式”（W.F.F.）和“转换”这些术语。在第二个证明中，我们假定了一些公式已经存在，不再证明。这些公式可以参考克莱尼的论文“形式逻辑的正整数理论”[发表在《美国数学杂志》57（1935），153-173，219-244]直接构造。

图灵这里说的“第二个证明”是指上述定理的逆命题：每一个可计算序列同样也是λ可定义的。

The W.F.F. representing an integer n will be denoted by N_n.

表示一个整数n的W.F.F.将记为N_n。

我们开始使用克莱尼对1和后继数字的定义，但要使这些自变量与图灵后面使用的保持一致，N_1是，$\lambda xy.x(y)$, N_2是$\lambda xy.x$（$x(y))$, N_n是$\lambda xy.x(x(x(x\ldots(y)\ldots)))$.

We shall say that a sequence γ whose n-th figure is $\varphi_\gamma(n)$ is λ-definable or effectively calculable if $1 + \varphi_\gamma(u)$ is a λ-definable function of n,

如果$1+\varphi_\gamma(u)$是n的λ可定义函数，我们就称第n个数字为$\varphi_\gamma(n)$的序列γ是λ可定义的或者是有效可计算的。

φ_r两次出现时的参数都应该是n，而不是u，因此表达式应该是$1+\varphi_\gamma(n)$。可计算序列γ的第n位是0或者1，但邱奇和克莱尼定义的λ演算

只考虑了正整数，并不包括0。$\varphi_\gamma(n)$并不是λ可定义的，因为0不是λ可定义的。所以，数字都加上1变成1和2。

i.e. if there is a W.F.F. M_γ such that, for all integers n,

$$\{M_\gamma\}(N_n) \text{ conv } N_{\varphi\gamma(n)+1},$$

i.e. $\{M_\gamma\}(N_n)$ is convertible into $\lambda xy.x(x(y))$ or into $\lambda xy.x(y)$ according as the n-th figure of λ is 1 or 0.

如果存在W.F.F. M_γ，使得对于所有的整数n，

$$\{M_\gamma\}(N_n)\text{conv}N_{\varphi_\gamma(n)+1},$$

即根据λ的第n位数是1还是0，$\{M_\gamma\}(N_n)$ 可以转换为$\lambda xy.x(x(y))$或者。$\lambda xy.x(y)$.

最后一行的λ其实是错的，应该是“γ的第n位数”。根据对应的数字是1还是0，值N_n（指序列中的数字位）处的函数M_γ可转换为 $\lambda xy.x(x(y))$（也就是2）或者 $\lambda xy.x(y)$（也就是1）。

例如，如果γ的第5位是1，那么$\varphi_{\gamma(5)}$是1，并且

$$\{M_\gamma\}(N_5) \text{ conv}N_{\varphi\gamma}(5)+1$$

也就是说

$$\{M_\gamma\}(N_5) \text{ conv}N_2$$

To show that every λ-definable sequence γ is computable, we have to show how to construct a machine to compute γ. For use with machines it is convenient to make a trivial modification in the calculus of conversion. This alteration consists in using $x, x', x'', \ldots$ as variables instead of $a, b, c, \ldots$.

为了证明每一个λ可定义序列γ都是可计算的，我们必须首先构造一个可以计算γ的机器。使用机器，可以方便地在演算转换中进行简单的修改，这些修改包括使用$x, x', x'', \ldots$ 作为变量，而不是使

用*a*, *b*, *c*….

图灵至今还未使用任何以*a*、*b*或*c*命名的变量，但他使用了*x*和*y*。他希望所有的变量都具有统一的形式，因为接下来需要进行比较和匹配。这与第8节的要求很相似（本书第205页）：一阶谓词逻辑在被机器处理前需要“系统化”。

We now construct a machine $\mathcal{L}$ which, when supplied with the formula M_γ, writes down the sequence *γ*. The construction of $\mathcal{L}$ is some-what similar to that of the machine $\mathcal{K}$ which proves all provable formulae of the functional calculus. We first construct a choice machine $\mathcal{L}_1$, which, if supplied with a W.F.F., *M* say, and suitably manipulated, obtains any formula into which *M* is convertible. $\mathcal{L}_1$ can then be modified so as to yield an automatic machine $\mathcal{L}_2$ which obtains successively all the formulae

[264]

into which *M* is convertible (cf. foot-note p. 252).

我们现在构造一台机器$\mathcal{L}$，在提供了公式M_γ时可以写下序列*γ*。$\mathcal{L}$的构造过程与$\mathcal{K}$在某种程度上很相似，$\mathcal{K}$证明了

函数演算中所有可证明的公式。我们首先构造一个选择机器$\mathcal{L}_1$，如果给$\mathcal{L}_1$提供了一个W.F.F.，例如M，并合理调整，使$\mathcal{L}_1$包含任一M可转换到的公式，则可以调整$\mathcal{L}_1$使之衍生出自动机器$\mathcal{L}_2$，$\mathcal{L}_2$相继得到所有M可转换到的公式（参见第252页脚注）。

在图灵论文的第252页有5个脚注，图灵引用的是第二个（本书第205页）。在那里，他讨论了可以证明所有一阶逻辑的可证明公式的机器。考虑到λ表达式转换的系统化方式，这台机器看起来与之相似，很可能更加简单。

The machine $\mathcal{L}$ includes $\mathcal{L}_2$ as a part. The motion of the machine $\mathcal{L}$ when supplied with the formula M_γ is divided into sections of which the *n*-th is devoted to finding the *n*-th figure of *γ*. The first stage in this *n*-th section is the formation of $\{M_\gamma\}(N_n)$. This formula is then supplied to the machine $\mathcal{L}_2$, which converts it successively into various other formulae. Each formula into which it is convertible eventually appears, and each, as it is found, is compared with

$$\lambda x\Big[\lambda x'\big[\{x\}\big(\{x\}(x')\big)\big]\Big], \quad i.e.\ N_2,$$

and with

$$\lambda x[\lambda x'[\{x\}(x')]], \quad i.e.\ N_1.$$

$\mathcal{L}_2$是机器$\mathcal{L}$的一部分。提供公式M_γ时，$\mathcal{L}$的操作可以划分为很多部分，其中第n部分用来寻找γ的第n位。第n部分的第一阶段是公式$\{M_\gamma\}(N_n)$。这个公式接下来提供给$\mathcal{L}_2$用来将公式连续地转换为其他公式。每一个可转换的公式都会最终出现，并与下列公式进行比较：

$$\lambda x\left[\lambda x'\left[\{x\}\left(\{x\}(x')\right)\right]\right]，\text{即 } N_2,$$

及$\lambda x\left[\lambda x'\left[\{x\}(x')\right]\right]$，即 N_1。

这就是数字2和1的冗长的表达式。为了实现一个可以转换λ表达式的机器，你必须在表示法中保持一致，而不包含语法捷径的方法实际上是最简单的。

If it is identical with the first of these, then the machine prints the figure 1 and the *n*-th section is finished. If it is identical with the second, then 0 is printed and the section is finished. It it is different from both, then the work of $\mathcal{L}_2$ is resumed. By hypothesis, $\{M_\gamma\}(N_n)$ is convertible into one of the formulae N_2 or N_1; consequently the *n*-th section will eventually be finished, *i.e.* the *n*-th figure of γ will eventually be written down.

如果与第一个完全一样，那么机器打印1，第n部分完成。如果与第二个相同，那么打印0，这部分也结束了。如果与这两个都不

一样，那么$\mathcal{L}_2$的工作就要重新开始。根据假设，$\{M_\gamma\}(N_n)$是可以转换为公式N_2或者N_1，因此第n部分最终是会结束的，也就是说，γ的第n位最终会写出来。

在开始更加复杂的变换证明之前，图灵空了一行：如何设计一个能包括特定机器运转情况的λ表达式。

To prove that every computable sequence γ is λ-definable, we must show how to find a formula M_γ such that, for all integers n,

$$\{M_\gamma\}(N_n) \text{ conv } N_{1+\phi_\gamma(n)}.$$

为了证明每一个可计算序列γ都是λ可定义的，我们必须先说明如何找到一个公式M_γ，对于所有的整数n，

$$\{M_\gamma\}(N_n)\text{conv}N_{1+\phi_\gamma(n)}$$

这个公式正好和之前那个一样，只是最后那个N的下标被重排了。现在，要做的不是描述一个操作λ函数的机器，而是定义一个模拟机器的λ函数。

Let $\mathcal{M}$ be a machine which computes γ and let us take some description of the complete configurations of $\mathcal{M}$ by means of numbers, *e.g.* we may take the D.N of the complete configuration as described in §6.

假设$\mathcal{M}$是一台计算γ的机器，我们来用数字描述一下$\mathcal{M}$的完全格局。正如§6中描述的，我们可以得到完全格局的描述数。

在下面的讨论中，我将会引用到“格局数”，这其实就是简单的连续整数0, 1, 2, 3等，这个数字随机器运转而递增。对于任何一个特定的机器，对每个格局数，存在一个与完全格局对应的描述数。这些是非常大的数字，包括了描述纸带上已打印数字及下一个m-格局的代码。

Let $\xi(n)$ be the D.N of the *n*-th complete configuration of $\mathcal{M}$.

令$\xi(n)$为$\mathcal{M}$的第*n*个完全格局的描述数。

图灵在这里使用的*n*是我所指的格局数，而$\xi(n)$是一个描述数。

The table for the machine $\mathcal{M}$ gives us a relation between $\xi(n+1)$ and $\xi(n)$ of the form

$$\xi(n+1) = \rho_\gamma\left(\xi(n)\right),$$

where ρ_γ is a function of very restricted, although not usually very simple, form: it is determined by the table for $\mathcal{M}$.

从$\mathcal{M}$的格局表中，我们可以得到$\xi(n+1)$及$\xi(n)$之间的如下关系：

$$\xi(n+1) = \rho_\gamma\left(\xi(n)\right)$$

其中ρ_γ是一个严格受限的函数，尽管通常不是很简单：其形式由的格局表决定。

ρ_γ函数将一个描述数转换为下一个描述数。一般来讲，输入是一个长的数字，输出是另一个长的数字。这个函数基本上必须在这个长输入数中，寻找一组对应于m-格局和扫描符的特定模式的数字，并基于格局表构造下一个完全格局，有可能包含一个新的打印符号，并改变m-格局和下一个扫描符。

图灵对于这个函数的“通常不是很简单”的描述切中目标。这个函数需要将描述数拆分为单个数位并验证它们。因为描述数是十进制数，所以这个函数可以通过先将数字除以10的幂，忽略分数部分，再除以另一个10的幂，保留余数，来抽取其中任意长度的部分。尽管ρ_γ函数毫无疑问是很复杂的，但它肯定是能被想到的。

ρ_γ is λ-definable (I omit the proof of this), *i.e.* there is a W.F.F.A_γ such that, for all integers *n*,

$$\{A_\gamma\}(N_{\xi(n)}) \text{ conv } N_{\xi(n+1)}.$$

ρ_γ是λ可定义的（我略去了这部分的证明），即存在一个合式公式A_γ，对于所有的整数n：

$$\left\{A_\gamma\right\}\left(N_{\xi(n)}\right)\text{conv}N_{\xi(n+1)}$$

A_γ本质上是与ρ_γ一样的函数，只不过是用λ演算的语言表示的。它用来进行描述数间的转换。

Let U stand for

$$\lambda u\left[\{\{u\}(A_\gamma)\}(N_r)\right],$$

where $r=\xi(0)$;

令U表示

$$\lambda u\left[\{\{u\}(A_\gamma)\}(N_r)\right],$$

其中$r=\xi(0)$;

这句开头的大写U应该有一个下标γ，因为它是基于一个特定可计算序列的。N_r是机器开始时的完全格局的描述数——数字313。这个数字对应于标准描述DAD，意思是m-格局q_1（DA）和扫描一个空格（D）。变量u是格局数，随着机器运行依次是0、1、2等。包含u的是个花括号，通常意味着是个函数，虽然看起来似乎没什么意义，但很快你就将看到它很有效。

then, for all integers n,

$$\{U_\gamma\}(N_n)\ \text{conv}\ N_{\xi(n)}.$$

那么，对于所有的整数n，

$$\{U_\gamma\}(N_n)\,\text{conv}N_{\xi(n)}$$

U_γ函数的参数是格局数（0，1，2等）。图灵认为，这个函数可以转换为那个格局的描述数。我们使用描述数4来验证一下，其中需要涉及转换表达式$\{U_\gamma\}(N_4)$，即：

$$\left\{\lambda u\left[\left\{\{u\}\left(A_\gamma\right)\right\}\left(N_{\xi(0)}\right)\right]\right\}\left(\lambda xy.x\left(x\left(x\left(x\left(y\right)\right)\right)\right)\right)$$

在U_γ函数中，我使用了$N_{\xi(0)}$而不是N_r，正因如此，我们不会忘记下标代表了描述数。用4的表达式替换函数中的u：

$$\left\{\left\{\lambda xy.x\left(x\left(x\left(x\left(y\right)\right)\right)\right)\right\}\left(A_\gamma\right)\right\}\left(N_{\xi(0)}\right)$$

现在将x替换为A_γ：

$$\left\{\lambda y.A_\gamma\left(A_\gamma\left(A_\gamma\left(A_\gamma\left(y\right)\right)\right)\right)\right\}\left(N_{\xi(0)}\right)$$

最后我们用$N_{\xi(0)}$替换y：

$$A_\gamma\left(A_\gamma\left(A_\gamma\left(A_\gamma\left(N_{\xi(0)}\right)\right)\right)\right)$$

A_γ在$N_{\xi(0)}$上第一个应用的结果是$N_{\xi(1)}$，下一次得到$N_{\xi(2)}$，等等，因此最后的结果就是$N_{\xi(4)}$，这和图灵预想的一样。现在，你就可以看出为什么将u定义为U_γ中的函数是有意义的了：它可以在根本上将出现在A_γ函数中的u混合嵌套。

It may be proved that there is a formula V such that

$$\left\{\{V\}(N_{\xi(n+1)})\right\}(N_{\xi(n)}) \begin{cases} \text{conv } N_1 & \text{if, in going from the } n\text{-th to the } (n+1)\text{-th complete configuration, the figure 0 is printed.} \\ \text{conv } N_2 & \text{if the figure 1 is printed.} \\ \text{conv } N_3 & \text{otherwise.} \end{cases}$$

可以证明，对于公式V，有

$$\left\{\{v\}\left(N_{\xi(n+1)}\right)\right\}\left(N_{\xi(n)}\right) \begin{cases} \text{conv } N_1 & \text{打印了0，从第}n\text{个完全格局转换至第}n+1\text{个完全格局，} \\ \text{conv } N_2 & \text{打印了1} \\ \text{conv } N_3 & \text{其他情况} \end{cases}$$

函数V基本上分析了连续两个完全格局的描述数，并确定打印0或者1，或者什么都不打印，这又是一个复杂但可以想到的函数。

Let W_γ stand for

$$\lambda u\left[\left\{\{V\}\left(\{A_\gamma\}\left(\{U_\gamma\}(u)\right)\right)\right\}\left(\{U_\gamma\}(u)\right)\right],$$

so that, for each integer *n*,

$$\left\{\{V\}(N_{\xi(n+1)})\right\}(N_{\xi(n)}) \text{ conv } \{W_\gamma\}(N_n),$$

令W_γ表示

$$\lambda u\left[\left\{\{V\}\left(\left\{A_\gamma\right\}\left(\left\{U_\gamma\right\}(u)\right)\right)\right\}\left(\left\{U_\gamma\right\}(u)\right)\right],$$

因此，对于任意一个整数n，

$$\left\{\{V\}\left(N_{\xi(n+1)}\right)\right\}\left(N_{\xi(n)}\right)\text{conv}\left\{W_\gamma\right\}\left(N_n\right),$$

这个命题左侧的公式可以转换为N_1、N_2或 N_3中的一个。描述这种变换比较容易，可以先描述转换后的结果：

$$\left\{W_\gamma\right\}\left(N_n\right)$$

将W_γ替换为图灵刚刚给出的那个表达式

$$\left\{\lambda u\left[\left\{\{V\}\left(\left\{A_\gamma\right\}\left(\left\{U_\gamma\right\}(u)\right)\right)\right\}\left(\left\{U_\gamma\right\}(u)\right)\right]\right\}\left(N_n\right)$$

用N_n代替u：

$$\left\{\{V\}\left(\left\{\mathrm{A}_\gamma\right\}\left(\left\{\mathrm{U}_\gamma\right\}\left(N_n\right)\right)\right)\right\}\left(\left\{\mathrm{U}_\gamma\right\}\left(N_n\right)\right)$$

表达式$\{U_\gamma\}(N_n)$可以转换为$N_{\xi(n)}$, 因此：

$$\left\{\{V\}\left(\left\{A_\gamma\right\}\left(N_{\xi(n)}\right)\right)\right\}\left(N_{\xi(n)}\right)$$

表达式$\{A_\gamma\}(N_{\xi(n)})$可以转换为$N_{\xi(n+1)}$，这正是我们所求的：

$$\left\{\{V\}\left(N_{\xi(n+1)}\right)\right\}\left(N_{\xi(n)}\right)$$

通过这个简短的证明，我们可以知道$\{W_\gamma\}(N_n)$可以转换为N_1、N_2或N_3，这取决于第n个完全格局转换为第n+1个时打印的是0还是1，或是什么都不打印。

and let Q be a formula such that

$$\left\{\{Q\}(W_\gamma)\right\}(N_s)\ \text{conv}\ N_{r(z)},$$

where $r(s)$ is the s-th integer q for which $\{W_\gamma\}(N_q)$ is convertible into either N_1 or N_2.

令Q是一个公式，使得

$$\Big\{\{Q\}\big(W_\gamma\big)\Big\}\big(N_s\big)\,\mathrm{conv}\,N_{r(z)}\ ,$$

其中$r(s)$是第s个整数q，其中$\{W_\gamma\}(N_q)$转换为N_1或N_2。

在上面的公式中，最后一个N的下标明显应该是$r(s)$而不是$r(z)$。只有一部分的完全格局才会打印0或者1，函数$r(s)$就是用来揭示这些完全格局的。例如，如果在1、4、6、7等完全格局中打印了0或者1，那么$r(1)$返回1，$r(2)$返回4，$r(3)$返回6，$r(4)$返回7，等等。

Then, if M_γ stands for

$$\lambda w\left[\{W_\gamma\}\left(\Big\{\{Q\}(W_\gamma)\Big\}(w)\right)\right],$$

it will have the required property[†].

[†] In a complete proof of the λ-definability of computable sequences it would be best to modify this method by replacing the numerical description of the complete configurations by a description which can be handled more easily with our apparatus. Let us choose certain integers to represent the symbols and the m-configurations of the machine. Suppose that in a certain complete configuration the numbers representing the successive symbols on the tape are $s_1 s_2 \ldots s_n$, that the m-th symbol is scanned, and that the m-configuration has the number t; then we may represent this complete configuration by the formula

$$\big[[N_{s_1}, N_{s_2}, \ldots, N_{s_{m-1}}], [N_t, N_{s_m}], [N_{s_{m+1}}, \ldots, N_{s_n}]\big],$$

where

$$[a,b] \text{ stands for } \lambda u\Big[\big\{\{u\}(a)\big\}(b)\Big],$$

$$[a,b,c] \text{ stands for } \lambda u\left[\left\{\big\{\{u\}(a)\big\}(b)\right\}(c)\right],$$

etc.

如果M_γ表示

$$\lambda w\left[\{W_\gamma\}\left(\{\{Q\}(W_\gamma)\}(w)\right)\right],$$

那么它将具备所需要的性质。[†]

[†] 在可计算序列的λ可定义性的完整证明中，最好改变这个方法，用我们的机器能够很好处理的描述代替完全格局的数字描述。我们选择某些整数来代表这些符号和机器的m-格局。假设在特定的完全格局中，那些代表纸带上后续符号的数字是$S_1S_{2\ldots}S_n$，其中第m个符号被扫描到，m-格局的标号为t，那么我们可以用下面的公式表示这个完全格局

$$\left[\left[N_{s_1},N_{s_2},\cdots,N_{s_{m-1}}\right],\left[N_t,N_{s_m}\right],\left[N_{s_{m+1}},\cdots,N_{s_n}\right]\right],$$

其中

$$[a,b]\ 代表\ \lambda u\left[\{\{u\}(a)\}(b)\right]$$

$$[a,b,c]\ 代表\ \lambda u\left[\{\{\{u\}(a)\}(b)\}(c)\right],$$

等等。

在图灵证明的第二部分，他开始自己寻找公式M_γ，使得对于所有n，

$$\{M_\gamma\}(N_n)\,\mathrm{conv}N_{1+\phi_\gamma(n)}$$

这个公式告诉我们序列的第n位是0还是1。我们先来分析：

$$\{M_\gamma\}(N_n)$$

用图灵刚刚为M_γ推导出的公式代替之：

$$\left\{\lambda w\left[\{W_\gamma\}\left(\{\{Q\}(W_\gamma)\}(w)\right)\right]\right\}(N_n)$$

用N_n替代w：

$$\{W_\gamma\}\left(\{\{Q\}(W_\gamma)\}(N_n)\right)$$

括号中的表达式可以转换为$N_{r(n)}$，因此

$$\{W_\gamma\}\left(N_{r(n)}\right)$$

已经表明，这个公式可以转换为N_1、N_2或 N_3，这主要取决于完全格局$r(n)$打印的是0或1或其他。然而，$r(n)$定义了只有在完全格局打印了0和1的情况下才返回。

上面的脚注表明，完全格局在纸带上被分为了下一个扫描符之前部分及下一个扫描符之后部分。图灵所建议的用来表示纸带这些部分的λ表达式可以是非常长，并且大小会随着完全格局的增长而增长。

下面是论文的结尾。

The Graduate College,
Princeton University,
New Jersey , U.S.A.

研究生院，
普林斯顿大学，
美国新泽西州。

对于更加严格的证明，图灵并没有遵循他在这里概述的证明逆命题的思路。论文“可计算性和λ可定义性”于1937年9月11日被《符号逻辑杂志》收录，这时距他抵达普林斯顿还不到一年的时间。[20]文章开始写道：

> 我们已经给出了许多定义用来表述与“有效计算性”的直观思路相对应的确切含义，并将其应用到了正整数函数中。这篇论文的意图是为了表明作者所介绍的可计算函数与邱奇发明的λ可定义的函数，以及与由赫尔布兰德和哥德尔引入并由克莱尼发展的一般递归函数是一致的。（在本论文中）已表明，任意的λ可定义函数都是可计算的，并且每一个可计算函数是一般递归函数。

图灵首先通过给出一台图灵机来证明λ可定义函数是可计算的。这台图灵机可能比图灵的通用机更复杂，它可以解析和转换λ函数。

证明的第二部分展示了可计算函数是递归的。图灵并不需要证明可计算函数是λ可定义的，因为斯蒂芬·克莱尼已经证明了递归函数是λ可定

义的（在“λ可定义性和递归”一文中）。这样，有效计算性的三个定义被等价地联系起来了。

图灵后来经常提到他1935年夏躺在格兰切斯特庄园草地上时构想出来的这些神奇的虚拟机器，但是在他后续发表的论文中并没有给出机器的实际格局表。当他在邱奇的指导下写博士论文时，[21]论文内容已经完全与递归函数和λ函数相关了。

---

[1] Maria Manzano，“Alonzo Church: His Life，His Work and Some of His Miracles”，*History and Philosophy of Logic*，Vol. 18（1997），212。

[2] P. J. Landin，“A Correspondence Between ALGOL 60 and Church’s Lambda-Notation”，*Communications of the ACM*，Vol. 8，No.2（Feb 1965），89-101，Vol. 8，No. 3（Mar 1965），158-165。

[3] Herbert B. Enderton，“Alonzo Church: Life and Work”，introduction to *Collected Works of Alonzo Church*（MIT Press，forthcoming）引言，http:// www.math.ucla.edu/~hbe/church.pdf。

[4] Herbert B. Enderton, “In Memoriam: Alonzo Church, 1903–1995,” *The Bulletin of Symbolic Logic*, Vol. 1, No. 4 (1995), 486–488。

[5] 阿隆佐·邱奇，“A Set of Postulates for the Foundation of Logic”，*The Annals of Mathematics*，2nd Series，Vol. 33，No. 2（Apr.1932），346-366。

[6] J. Barkley Rosser，“Highlights of the History of Lambda-Calculus”，*Annals of the History of Computing*，Vol. 6，No. 4（Oct.1984），337-349。

[7] 邱奇，“A Set of Postulates for the Foundation of Logic（Second Paper）”，*The Annals of Mathematics*, 2nd Series，Vol. 34，No. 4（Oct. 1933），839-864。

[8] 邱奇，842。

[9] 邱奇，864。

[10] S. C. 克莱尼，“Proof by Cases in Formal Logic”，*The Annals of Mathematics*， 2nd Series，Vol. 35，No. 3（July 1934），529-544。

[11] S. C. 克莱尼，“A Theory of Positive Integers in Formal Logic，Part I”，*American Journal of Mathematics*，Vol. 57，No. 1（Jan. 1935），153-173；Vol. 57，No. 2（Apr. 1935），219-244。

[12] 邱奇和罗瑟，“Some Properties of Conversion”，*Transactions of the American Mathematical Society*，Vol. 39，No. 3（May 1936），472-482。

[13] 我将采用克莱尼的 “A Theory of Positive Integers in Formal Logic,

Part I” 中前10页出现的定义。

[14] 克莱尼，“Origins of Recursive Function Theory”，*Annals of the History of Computing*，Vol. 3，No. 1（Jan 1981），59。

[15] 根据克莱尼和罗瑟的笔录，哥德尔的讲义早已在内部流传，但直到1965年才正式出版在马丁·戴维斯，ed.，*The Undecidable*（Raven Press，1965），41-71中。接着它们又被出版在库尔特·哥德尔，*Selected Works: Volume I, Publications* 1929-1936（Oxford University Press，1986），346-371。

[16] 邱奇，“An Unsolvable Problem of Elementary Number Theory”，*American Journal of Mathematics*，Vol. 58，No.2（Apr. 1936），345-363。

[17] 克莱尼，“General Recursive Functions of Natural Numbers”，*Mathematische Annalen*，Vol. 112，No.1（Dec.1936），727-742，重印于马丁·戴维斯，ed.，*The Undecidable*（Raven Press，1965），237-252。S. C. 克莱尼，“λ-Definability and Recursiveness”，*Duke Mathematical Journal*，Volume 2，Number 2（1936），340-353。

[18] 邱奇，“A Note on the Entscheidungsproblem”，*The Journal of Symbolic Logic*，Vol. 1，No. 1（Mar. 1936），40-41。也可参见邱奇，“Correction to a Note on the Entscheidungsproblem”，*The Journal of Symbolic Logic*，Vol. 1，No. 3（Sep. 1936），101-102。

[19] 安德鲁·霍奇斯，*Alan Turing: The Enigma*（Simon & Schuster，1983），116。

[20] 阿兰·图灵，“Computability and λ-Definability”，*The Journal of Symbolic Logic*，Vol. 2，No. 4（Dec. 1937），pp.153-163。

[21] 阿兰·图灵，“Systems of Logic Based on Ordinals”，*Proceedings of the London Mathematical Society*，2nd Series，Vol. 45，No.1（1939），161-228。

第16章 对连续统的设想

历史总是试图用一系列连贯的语句和段落来捕捉生活，然而现实生活通常杂乱且复杂得多。历史学家必须磨平事实的粗糙棱角，忽略次要人物，以避免离题。这些简化有时候会扭曲它试图阐述的事物，导致一系列看上去并不自然却又不可避免的事件发生，好像任何事情都不能改变它们的发展，甚至暗示着这些事件就是所有可能中最好的结果。这些扭曲的结果有时候会称为所谓的"历史的辉格解释"——19世纪的那些作家把大英帝国的历史描绘成正在逐步、无情地走进现代议会民主制，在此之后，英国的历史学家赫伯特·巴特菲尔德（1900—1979）称之为"历史的辉格解释"。

科学史、数学史和技术史尤其容易受到"辉格解释"的影响。我们是"正确的"科学理论和"适当的"技术的受益者，因此，我们能够找到贯通历史的链条，把结果与导致其不可避免发生的原因联系起来。探索过程中的过失都被忽略了，而如果讨论的是历史争论，任何阻止我们通向即将为之庆祝的辉煌时刻的人都将被征服。

例如，在有关图灵机的历史中，人们努力将它看做一系列连贯的、逐步实现的智慧成就，从康托尔和弗雷格，到罗素和希尔伯特，再经过哥德尔、邱奇和图灵，最终在1936年发表的一篇数学论文后达到顶点。为了保持这本书内容的合理精简并且突出重点，书中也正是这么描述的。

在这个进展中，我忽略了一些异议。如同任何智力劳动领域一样，数学史上也充斥着争论与不合。[1]在19世纪末期和整个20世纪，这些争论频繁地涉及数学哲学，特别是经常涉及无穷的本质。

数学哲学是一个广泛而又复杂的领域，不过，或许最基本的问题既简单又令人不安，那就是：

> 数学实体在何种程度上独立于研究它的人类。

难道数学家们只是采用与天文学家发现恒星和其他天体差不多的方

式，来发现已经存在于宇宙固有结构中的数学模式吗？抑或数学家们就像工程师设计一个新的真空吸尘器，或者作曲家写一部歌剧一样来发明数学吗？伟大的数学史普及者莫里斯·克莱因（1908—1992）这样诗意地描述人们关于数学本质的疑问：

> 数学是那些埋藏在宇宙深处，被逐渐发掘出来的钻石吗？或者它就像人类生产的人造宝石，尽管灿烂，然而也让那些发明它们的数学家被盲目的骄傲所蒙蔽？[2]

争论的一方是现实主义者或者柏拉图主义者，用罗杰·彭罗斯的话说："他们相信数学真理的客观性。依我看来，柏拉图式的存在是指一个客观的外部标准的存在，它不依赖于我们个人的思想和某种特定文化。"[3]

另一极端来自构成主义者，他们认为数学完全是由人类发明的。对构成主义者来说，数学表面上的永恒和超然存在仅仅是被人类模式识别技能加强的一种错觉，这种技能是在我们大脑中经过数百万年进化形成的。

这两个极端之间还有很多等级，每个级别都有各自的描述名称和拥护者，他们当中很可能已有部分人对我这种粗糙地在两个极端间进行分类不满了。

大多数数学工作者会把自己归类于柏拉图主义者。对数学的柏拉图式的理念支配着我们的文化，吸引着我们的本能。当我们大喊"找到了！"的时候，是表示"我发现了它"，而不是"我发明了它"。大卫·希尔伯特在第二届国际数学家大会上发表演讲之后的100多年来，我们仍然会对他的话语感到激动不已。他说：

> 无论这些问题看似如何地难以接近，无论我们站在它们面前感觉有多么无助，我们都坚信，经过有限步纯粹的逻辑过程之后，我们必定可以找到答案……这种对于任何数学问题都可解的坚信，对数学工作者是一种强有力的鼓励。我们听到内心不停地呼唤：存在一个问题，探索它的答案。你可以通过纯粹的推理找到答案，因为在数学里没有人类不可知的东西。[4]

这就是柏拉图主义者的演讲，他们认为解决方法已经存在，我们只需要去寻找它们。甚至在希尔伯特关于证明完备性、一致性和判定过程的希望破灭后，柏拉图主义者的直觉仍然幸存。事实上，柏拉图主义者当中比较杰出的是库尔特·哥德尔。

数学史中的分歧不仅仅是意识形态的问题，同时也集中在适当性

上。某些基本的假设构成所有数学证明的基础。然而，有一些假设是在有限对象的条件下提出的，在将其应用到无穷集合时就会出问题。

“对无穷的思考并非一帆风顺，”亚里士多德（公元前384—前322年）觉察到，我们可以想象他的学生郑重地点头表示同意，”无论你假设是否存在无穷这种事物，都会得到很多棘手的结论。”[5]

在亚里士多德的《物理》（第三册）中，为了探究这个变幻莫测的问题，他区分了实无穷和潜无穷，这样做是很有好处的。潜无穷是自然数的无穷：每个数后面都紧跟着另一个数。把一个事物细分成越来越小的碎片也是潜无穷。存在着某些随着时间不断发生，并且永远不会结束的过程。“一般说来，事情接踵而至的时候便产生了无穷。正在发生的事件本身是有限的，但在之后总是紧接着出现与其不同的另一事件。”[6]

然而，在亚里士多德的宇宙论中不存在实无穷，他提出几种论证解释为什么不存在实无穷。他聪明地注释道：“事实上，无穷与人们所理解的截然相反。它不是指’除本身之外没有任何事物了’，而是’除本身之外总是存在其他一些事物’。”[7]

亚里士多德甚至不允许无穷作为一个意识层面的概念而存在：

> 依赖于人类大脑所想到的东西是很荒谬的，因为过度和缺陷只会存在于大脑中，而在实际世界中是不存在的。可以把我们当中的任何一个人想象得比实际大很多倍，让他变得无穷大，但是这个人并不会因为其他人这样想而变得超出常人那么大。他只能和实际一样大，如果别人想象着他也是这样大，那么只是巧合而已。[8]

亚里士多德不是柏拉图主义者。

不是每个人都会接受亚里士多德拒绝无穷的观点。哲学家斯蒂芬·克尔纳（1913—2000）这样评论亚里士多德的理论：

> 它们从来不会被无异议地接受。柏拉图主义的哲学家们，包括奥古斯丁的神学家，总是将已知总体的无穷概念视为合法的，无论它们连续与否。他们不会因这个概念不适用于感官体验而苦恼，因为他们认为数学不是感官体验的抽象，更不是感官体验的描述，而是对事实的描述。事实不能通过感官理解，而要靠推理来认识。[9]

数学家们经常被实无穷困扰着，并且试图以一种安全的方式来处理无穷过程。正是认识到实无穷和潜无穷间的区别，使得我们才能得到如下的公式

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

而不是

$$\left(1+\frac{1}{\infty}\right)^\infty$$

第一个公式表示了一个极限。当n的值越来越大的时候，这个表达式的值趋向于何值呢？它向我们熟知的欧拉常数靠近，约等于2.71828…。

第二个公式采用符号∞表示一个实无穷，因此根本无意义。

尽管早期的数学家们已经接近极限的定义了，但数学上极限的严格定义是由德国数学家魏尔斯特拉斯（1815—1897）提出来的。这个概念对于微积分学是至关重要的，它为微分和积分运算提供了坚实的数学基础。在极限的概念提出之前，微积分学是基于“无穷小”概念的，也就是一个非零数（因为仍可以将其视为一个有限量来操作），但是又足够接近零，使得最终可以忽略它。微积分学中仍然存在“无穷小”，例如采用dx表示这些无穷小量。

19世纪的另一位德国数学家利奥波德·克罗内克（1823—1891）极其赞赏实无穷在数学中的应用。克罗内克以其“上帝创造了整数，其他一切都是人造的”[10]格言而闻名。提到上帝——或者更准确地说，对独立于人类而存在的数学实体的认同，使得克罗内克看上去像是个柏拉图主义者，但是从“其他一切”可以看出，他是一个绝对的构成主义者。克罗内克想把数学中的一切都建立在有限整数的有限结构的基础上，他研究的问题甚至还包括极限的概念和无理数的定义。

克罗内克从前的一位学生开始从事一项令人吃惊的数学研究——他不仅定义了无穷对象的集合，还对这些无穷对象进行了杂乱的计数，然后对这些值进行了算术运算。克罗内克反对这些方法，甚至一度阻止发表这些研究结果，导致了克罗内克对这位学生进行邪恶发狂的迫害的名声。这名学生就是格奥尔格·康托尔。

认同克罗内克的看法的人认为这种迫害更多地应该归于康托尔偏执的世界观，而非克罗内克的真实意图。[11]然而，历史是由胜利者书写的。康托尔的集合论及其对可数集合和不可数集合的区分，后来被证明是非常有用的，因此数学史基本上抛弃了克罗内克的理论。

康托尔提出的超限数的概念是极其柏拉图主义的，甚至有点让人摸不着头脑。下面的这些话是他在1883年写下的：

> 我们能够从两种意义上——无论是有限还是无限——谈论整数

> 的现实性或者存在性……首先，我们可以认为整数是实际存在的，因为基于目前为止的所有定义，我们完全都是这样理解的……但是，如果把数字看作是一个表达式，或者是对与智力相对的外部世界中事件和关系的一种复制，那么现实可以认为是由数字构成的……同时，因为彻底的现实性，我的观点也有些理想主义基础，那么在前一方面中存在的概念总是拥有以某种甚至无限种方式存在着的短暂现实，在这种意义上，我确定这两类现实总是同时发生的……这两种现实之间的关联基于我们所归属的一切的统一。[12]

我们在前面的章节中讨论了伯特兰·罗素的逻辑主义（来自弗雷格与皮诺亚）和大卫·希尔伯特的形式主义。在20世纪早期，与它们相对的另一种运动和哲学也发展起来了，这就是直觉主义，它是由荷兰数学家鲁伊兹·艾格博特斯·杨·布劳威尔（1881—1966）提出来的。

布劳威尔是一个忧郁悲观而又神秘古怪的人，生活于20世纪初，他就像一个惊骇于所感受到的混乱的严厉校长。研究布劳威尔的学者沃尔特把布劳威尔的人生观描述为“浪漫的悲观主义和彻底的个人主义的混合体”。在早期一本名为《生活，艺术和神秘主义》（1905）的专著中，布劳威尔“痛骂工业污染和人类借用其智慧及所建立起的社会结构来主宰自然的行径，并提倡回归’自然的’、神秘而静寂的深思”。[13]

布劳威尔考入阿姆斯特丹大学，后来在那里任教。尽管他的博士论文是关于数学基础（预示着他后来的兴趣）的，但是他的大部分早期工作都属于拓扑学领域。

布劳威尔创造了术语“直觉主义”来描述他关于如何用主观思想形式化数学实体的想法。它们都是思考的对象，它们在纸上的符号表示对于把思想从一个人表达给另一个人并无好处。相反，形式主义更注重对出现在整张纸中的符号的操作——这种做法就比一个无意义的规则下的游戏略好一点。

随着罗素和希尔伯特的纲领的逐渐形成，康托尔的工作在当时也已经被广泛接受。从布劳威尔的观点（亨利·庞加莱也持有相同的看法）来看，康托尔的集合论和超限数的广泛应用只能导致数学的灾难。

布劳威尔并不完全反对无穷的概念，他接受无穷集的构想，但前提是这些集合是可构造并且可枚举的；也就是说，它们可以一对一地与整数对应。早在1913年，布劳威尔就强调：“直觉主义只认识到可数集的存在……并且他们认为，阿列夫零是唯一存在的无穷幂”。[14]实数集合必须被严格禁止，因为这样的集合里的元素是不可数的。定义实数集合的唯一方法是声明这种集合包含了所有的实数。你不能只展示前面的几个数，然后加一个省略号，或者定义某种包含规则。没有规则，也没

有序列，你不能构造这个集合，所以也不存在这样的集合。因此，康托尔关于超限数的大部分理论“对直觉主义者毫无意义”。[15]

1918年到1928年期间，布劳威尔发表了关于直觉论者批判形式主义纲领的文章，同时发表了一些尝试为数学提供一个摆脱问题和悖论的新基础的文章。特别地，他发现了排中律的问题。排中律是确定某一事物拥有某一个属性或者不拥有该属性，而不存在其他情况的原理。这个定律适用于有限集合，布劳威尔认为把它应用到无穷集合上是愚蠢的。

在一个著名的例子中，[16] 布劳威尔也相信一个收敛序列的极限总是小于零，或大于零，或等于零。（从某个意义上，这与排中律是相关的，根据排中律，这个极限或者小于零，或者不小于零。）

下面是一个数列的定义：

$$C_n=(-\frac{1}{2})^n \quad n<k$$

$$C_n=(-\frac{1}{2})^k \quad n\geqslant k$$

这个数列的前几个值依次是 $-\frac{1}{2},\frac{1}{4},-\frac{1}{8},\frac{1}{16},-\frac{1}{32}$，，因此当$n$小于$k$时，这个数列显然是收敛于零的。并且，当$n$大于$k$时，剩余的所有$c_n$值均为$\left(-\frac{1}{2}\right)^k$，因此这即为该数列所收敛的值。

问题在于：k的值是连续数字0123456789在π在中首次出现的位置。

c_n会收敛于一个小于零的值，或者收敛于一个大于零的值，或者收敛于零吗？这取决于k是一个奇数，还是偶数，或者二者都不是。柏拉图主义者认为，序列c_n的极限是一个实实在在的数，哪怕我们不知道这个数是什么。构成主义者可能会反击说，因为这个极限是不能被构造出来的，因此它是不存在的。它是未定义的，排中律在此便不适用了。

有一次，布劳威尔在演讲关于这一不定序列的问题，有人指出，虽然我们可能不知道c是怎么收敛的，但上帝知道。布劳威尔回答道：“我可没法同上帝说话。”[17]

我们现在已经知道布劳威尔的序列是收敛的，虽然这个事实仅仅在他死后30年就为我们所知。[18]连续的0123456789出现在π出的第17387594880位，所以c_n收敛于$2^{-17387594880}$。既然布劳威尔原来的序列在

现在已经荒废，我们很自然会想到*k*的另一种判别方法。我们重新定义*k*为π的1百万个连续的7出现的位置。因为π出看上去是以一种较均匀的方式分布在各个数位上的，所以这1百万个连续的7也许不会出现在某个地方。（也许会。虽然很多数学家相信π中能找到任何可能的数字序列，但这个观点从来没被证明。除非存在大于宇宙的计算资源，否则无法在π大中找到。）

作为拒绝无穷集合排中律的一个结果，布劳威尔还否定了一些归谬法证明的合理性，以及希尔伯特主张的任何数学问题都是可以证明的观点！

逻辑的世界同样也受到影响。排中律可以用命题逻辑表示为：

$$-X \vee X$$

在怀特海、罗素和希尔伯特的经典逻辑体系里，这个算式等同于：

$$X \rightarrow X$$

而它们都等价于：

$$-(X \,\&\, -X)$$

蕴涵关系$X \rightarrow X$是古老哲学思想里的同一性（“某个事物是它自己”）的符号表示，而三个算式中最后那个则用符号表示了矛盾原理：某个事物不可能既具备某个属性而又没有那个属性。在亚里士多德逻辑学里，上面的算式代表的是不同的概念，但在命题逻辑这个“笨重的工具”的作用下，它们变成了同义词。[19]

对希尔伯特而言，布劳威尔希望强加给数学思想的”紧箍咒”过于严厉了。希尔伯特非常不愿意放弃他的数学工具，即使他承认一些情况必须重新考虑。“我们应该小心地审视那些形成观念的方式和带来丰富多彩的结论的推理模式；我们将看护它们，支持它们，让它们更为有用，无论成功的希望有多么渺小，没有人能将我们从康托尔为我们建立的天堂里驱逐出去。”[20]希尔伯特依然在寻找不需要使用到无穷的更为严格的证明方法。

希尔伯特和布劳威尔之间的流言蜚语渐渐升级到了顶点。1928年，布劳威尔被希尔伯特排挤出德国《数学年刊》（*Mathematische Annalen*）的编委位置。作为三个主编之一，爱因斯坦辞去了在《数学年刊》的职务，以此表示抗议。这次事件让布劳威尔感到万分苦闷和心灰意冷，此后十年他几乎再没有发表过论文。他在家门口被汽车撞到后不久去世，享年85岁。

我们不清楚图灵在动笔写他关于可计算数的论文前是否接触过直觉主义的思想。纽曼，这个给图灵上过数学基础课程并指导图灵发表论文

的剑桥大学教授，肯定很了解布劳威尔，因为他们彼此在拓扑学上合作过。纽曼共同起草了皇家学院为布劳威尔的官方讣告[21]（这不是一个普通的讣告，它有30页长，包括了5页对布劳威尔工作成果的介绍）。不过，纽曼只写了讣告中布劳威尔关于拓扑学研究成果的那一部分，而这也已经是在图灵的文章发表30年后的事情了。

图灵的文章很奇怪地处于形式主义和构造主义之间的孤岛上。他的机器虽然将算法限制到一系列事先定义了操作的打印符号上，但他区别实数和可计算数的方法，以及将可计算数定义为实数中那些确实可被计算的数的子集的方法，却着实有着构造主义的味道。图灵对于可计算数的定义后来导致了一门数学理论的诞生，那就是与经典的“实分析”（real analysis）理论相似的“可计算分析”（computable analysis）。[22]

和布劳威尔的思想相似的是，计算一个数是一个随时间发生的过程。在机器还没计算到数的某一位时这一位并不存在，图灵机也不能被一个有穷的通用过程判定出可能会在未来做什么。不存在一个算法，能让你从机器的描述数中判定出机器是否会打印0或1，或者它是否会打印有限个0和1，又或者它是否会打印出连续的0123456789。如果有这样的算法，我们就可以将它应用到机器中来计算π的无穷数位，我们就可以判定机器是否会打印连续的0123456789（或者1百万个连续的7），我们就会知道布劳威尔的序列是否会收敛于0。

如果存在这样的算法，那就意味着如和π其他无理数的那些无穷数位是一种独立的柏拉图式的存在，它们不需要实际计算就是存在的。但是，这样的算法并不存在，所以我们不得不艰难地寻找以获悉每一位数字是什么。

就像你所看到的，图灵有些时候喜欢定义某些数，这些数的数位需要依赖其他机器的分析结果，这些数最后都被发现是不可计算的。布劳威尔在他1921年的论文“每个实数都有一个十进制展开吗？”（Does Every Real Number Have a Decimal Expansion?）[23]中做了类似的事情。他定义了一个实数，它的每一位数字基于π的无穷数位上的某些数字模式出现的位置。

除了这些有趣的相似外，我在图灵1936年提交给伦敦数学学会的这篇论文中没有发现其他与布劳威尔直觉主义相似的证据。图灵的研究工作和结论是不同寻常的，我认为他并不是在某种特定数学哲学观点的条条框框下展开研究的。

1936年秋，图灵去了普林斯顿大学，和邱奇一起进行研究，随后逐渐表露出一些对数学可能性和思想的展望，其中可能包含了布劳威尔的直觉主义。

邱奇显然同直觉主义打过交道。当他1927年从普林斯顿拿到博士学位后，他曾经有两年的时间为国家研究奖学金资助的项目工作。

> 我在哈佛待了一年，在欧洲也待了一年。其中半年在哥廷根，因为希尔伯特当时在那里；半年在阿姆斯特丹，因为我对布劳威尔的研究很有兴趣，他的部分研究工作启发了我……我想他已经不再教书了，他显得很老。我常常得坐火车去他的住所，因为他的住所在很偏远的地方。[24]

“显得很老”这样的描述是比较反常的：邱奇接受这段采访的时候已经80岁，而1929年布劳威尔才48岁。也许布劳威尔和希尔伯特多年前的纷争确实对他打击很大。

随后，邱奇似乎有一种考虑直觉主义的倾向（尽管不是非常坚定的直觉主义论调）。邱奇的第一篇关于λ演算的论文开头一句是：“在这篇文章里，我们为形式逻辑提出了一个公理集合，其中避免使用了自由变量即实变量，我们也引入了对排中律的某种约束，这是为了避免与超限数学有关的矛盾”。[25]

邱奇的学生克莱尼的研究工作里也展示出其对直觉主义的兴趣。克莱尼在他的《数学导论》（*Introduction to Metamathematics*，1952）和稍后与人合作的《直觉主义数学的基础》（*The Foundations of Intuitionistic Mathematics*，1965）中，都包含了直觉主义的章节。一幅1953年布劳威尔的照片——摄于威斯康辛的麦迪逊，当时克莱尼在那里教书——出现在科林一篇关于递归函数理论发展历史的论文中。[26]

图灵同时可能也受赫尔曼·外尔的影响。外尔当时在高等研究院工作。外尔在希尔伯特的指导下在哥廷根完成博士学位，在苏黎世大学教过书，并于1930年返回哥廷根继承希尔伯特的衣钵，但在1933年因为妻子是犹太人而被迫离开德国。在1919年和1928年之间，外尔一直从直觉主义的角度研究数学，从来没有失去兴趣。

图灵一次简短的从直觉主义思想出发进行的尝试出现在他在普林斯顿时发表的一篇短文里。这篇短文更正了他先前那篇可计算数论文的一些观点。我在第53页讲过，那篇可计算数论文最初发表在伦敦数学会集刊第42卷第三部分（1936年11月20日发表）和第四部分（1936年12月23日发表）。那些发表于1936年10月和1937年4月之间的部分，后来被统一作为第42卷第2系列发表。

而图灵的这篇短文出现在伦敦数学会集刊第43卷第七部分（1937年12月30日发表）。它随后又被收入第43卷的第2系列，该系列包含了从1937年5月到12月发表的文章。

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM. A CORRECTION

By A. M. Turing.

In a paper entitled "On computable numbers, with an application to the Entscheidungsproblem"* the author gave a proof of the insolubility of the Entscheidungsproblem of the "engere Funktionenkalkül". This proof contained some formal errors† which will be corrected here: there are also some other statements is the same paper which should be modified, although they are not actually false as they stand.

---

* *Proc. London Math. Soc.* (2), 42(1936-7), 230-265.

† The author is indebted to P. Bernays for pointing out these errors.

《论可计算数及其在判定性问题上的应用》的更正

A. M. 图灵

在题为《论可计算数及其在判定性问题上的应用*》的论文中，作者给出了"狭义函数演算"（engere Funktionenkalkül）下判定性问题不可解的证明。这个证明包含了一些形式上的错误†，我将在这里更正。另外，那篇文章中有几个语句的表述也需要修改，虽然它们要表达的语义并没有错误。

---

* 伦敦数学会集刊，(2) 42 (1936-7), 230-265。

† 作者感谢贝奈斯指出了这些错误。

这个3页的短文分成了两部分。第一部分对原论文第11节判定性问题不可解的证明中一些公式和表述进行了更正。我已经在第14章讲述了这些更正。为了保持完整性，以下是该短文的一部分，我只会从中打断两次。

The expression for Inst $\{_{q_i}S_jS_kL_{q_l}\}$ on p.260 of the paper quoted should read

$$(x,y,x',y')\Big\{\Big(R_{S_j}(x,y)\ \&\ I(x,y)\ \&\ K_{q_i}(x)\ \&\ F(x,x')\ \&\ F(y',y)\Big)$$

$$\to\Big(I(x',y')\ \&\ R_{S_k}(x',y)\ \&\ K_{q_l}(x')\ \&\ F(y',z)\ \mathrm{v}\ \Big[\big(R_{S_0}(x,z)\to R_{S_0}(x',z)\big)$$

$$\&\ \big(R_{S_1}(x,z)\to R_{S_1}(x',z)\big)\ \&\ \dots\ \&\ \big(R_{S_M}(x,z)\to R_{S_M}(x',z)\big)\Big]\Big)\Big\},$$

$S_0,S_1,\dots,S_M$ being the symbols which $\mathcal{M}$ can print.

那篇论文第260页的表达式 Inst$\{q_iS_jS_kLq_l\}$应该这样解释：

$$(x,y,x',y')\Big\{\Big(R_{s_j}(x,y)\&I(x,y)\&K_{q_i}(x)\&F(x,x')\&F(y',y)\Big)$$

$$\to\Big(I(x',y')\&R_{s_k}(x',y)\&K_{q_l}(x')\&F(y',z)\mathrm{v}\Big[\big(R_{S_0}(x,z)\to R_{S_0}(x',z)\big)$$

$$\&\big(R_{S_1}(x,z)\to R_{S_1}(x',z)\big)\&\dots\&\big(R_{S_M}(x,z)\to Rs_M(x',z)\big)\Big]\Big)\Big\},$$

S_0, S_1,…, S_M是$\mathcal{M}$可以打印的符号。

这个更正其实也不怎么对。第二行的$F(y',z$前的z少了一个全称量词，这个全称量词修饰整个算式的余下部分。本书第250页的版本已经纠正了这个错误。

The statement on p. 261, line 33, viz.

$$\text{“Inst}\{q_a\ S_b\ S_d\ L\ q_c\}\ \&\ F^{(n+1)}\to(CC_n\to CC_{n+1})$$

is provable″ is false (even with the new expression for Inst $\{q_aS_bS_dLq_c\}$):

we are unable for example to deduce $F^{(n+1)} \rightarrow (—F(u,u''))$ and therefore can never use the term

$$F(y',z) \text{ v } \left[\left(R_{S_0}(x,z) \rightarrow R_{S_0}(x',z)\right) \& \ldots \& \left(R_{S_M}(x,z) \rightarrow R_{S_M}(x',z)\right) \right]$$

[545]

in Inst $\{q_a S_b S_d L q_c\}$.

第261页第33行的表述：

$$\text{“Inst}\left\{q_a S_b S_d L q_c\right\} \& F^{(n+1)} \rightarrow \left(CC_n \rightarrow CC_{n+1}\right)$$

是可证明的”是错误的（就算换成新的表达式Inst$\{q_a S_b S_d L q_c\}$也是错误的）：我们不能推导出$F^{(n+1)} \rightarrow (—F(u,u''))$，因此不能在Inst$\{q_a S_b S_d L q_c\}$中使用

$$F\left(y',z\right) \text{v}\left[\left(R_{S_0}\left(x,z\right) \rightarrow R_{S_0}\left(x',z\right)\right) \& \ldots \& \left(R_{S_M}\left(x,z\right) \rightarrow R_{S_M}\left(x',z\right)\right)\right]$$

这一项。

这就是图灵承认他对自然数的构建存在错误的地方。

To correct this we introduce a new functional variable G [$G(x, y)$ to have the interpretation “x precedes y'']. Then, if Q is an abbreviation for

$$(x)(\exists w)(y,z)\left\{F(x,w) \& \left(F(x,y) \rightarrow G(x,y)\right) \& \left(F(x,z) \& G(z,y) \rightarrow G(x,y)\right)\right.$$

$$\left. \& \left[G(z,x) \text{ v } \left(G(x,y) \& F(y,z)\right) \text{ v } \left(F(x,y) \& F(z,y)\right) \rightarrow \left(-F(x,z)\right)\right]\right\}$$

the corrected formula Un($\mathcal{M}$) is to be

$$(\exists u)A(\mathcal{M}) \to (\exists s)(\exists t)\, R_{S_1}(s,t),$$

where A($\mathcal{M}$) is an abbreviation for

$$Q \;\&\; (y)R_{S_0}(u,y) \;\&\; I(u,u) \;\&\; K_{q_1}(u) \;\&\; \mathrm{Des}(\mathcal{M}).$$

The statement on page 261 (line 33) must then read

$$\mathrm{Inst}\{q_a\, S_b\, S_d\, Lq_c\} \;\&\; Q \;\&\; F^{(n+1)} \to (CC_n \to CC_{n+1}),$$

and line 29 should read

$$r\left(n,\, i(n)\right) = b, \quad r\left(n+1,\, i(n)\right) = d, \quad k(n) = a, \quad k(n+1) = c.$$

For the words “logical sum” on p.260, line 15, read “conjunction”. With these modifications the proof is correct. Un($\mathcal{M}$) may be put in the form (I)(p.263) with *n*=4.

为了更正这个错误，我们引入了一个新的函数变量*G* [*G*(*x*，*y*), 表示“*x*在*y*之前”]。于是，如果*Q*表示

$$(x)(\exists w)(y,z)\{F(x,w)\&(F(x,y)\to G(x,y))\&(F(x,z)\&G(z,y)\to G(x,y))$$
$$\&[G(z,x)\vee(G(x,y)\&F(y,z))\vee(F(x,y)\&F(z,y))\to(-F(x,z))]\}$$

那么更正后的算式Un($\mathcal{M}$)就是

$$(\exists u)A(\mathcal{M})\to(\exists s)(\exists t)R_{S_1}(s,t),$$

其中，*A*($\mathcal{M}$)表示

$$Q\&(y)R_{S_o}(u,y)\&I(u,u)\&Kq_1(u)\&\mathrm{Des}(\mathcal{M}).$$

第261页第33行的句子应该解释为

$$\text{Inst}\{q_a S_b S_d L q_c\} \,\&\, Q \,\&\, F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1}) \text{ ,}$$

第29行的句子应该解释为

$$r(n,i(n)) = b, \quad r(n+1,i(n)) = d, \; k(n) = a, \quad k(n+1) = c \,.$$

第260页的词“逻辑和”应该更正为“合取”。通过这些更正，这个证明将是正确的。Un($\mathcal{M}$) 也许应该在n=4时变形为第263页I的形式。

更正的第一部分结束后是一个空行。第二部分是关于另一内容的，涉及原文中很早出现的一个段落。

Some difficulty arises from the particular manner in which “computable number” was defined (p.233).

用某一特定方法去定义“可计算数”会引发一些困难（第233页）。

相关的这一段在本书第68页：“如果一个序列可以被非循环机计算出来，那么它就是可计算序列。如果一个数与非循环机计算出来的数只相差一个整数，那么它就是可计算数。”

图灵在下一句话中的“直觉”一定指的是它的原意。如果图灵想表达任何同布劳威尔有关的意思，他应该会使用“直觉主义”一词。此外，从后面一句话中显然可以看出这个假设并不满足直觉主义的条件，即使它符合直觉认识。

If the computable numbers are to satisfy intuitive requirements we should have:

If we can give a rule which associates with each positive integer n two rationals a_n, b_n *satisfying* $a_n \le a_{n+1} < b_{n+1} \le b_n$, $b_n - a_n < 2^{-n}$, *then there is* a computable number α *for which* $a_n \le a \le b_n$ *each n.*

(A)

如果可计算数满足我们的直觉要求，那么有：

如果我们可以给出一个规则，对于每一个正整数n，都有两个有理数a_n和b_n，满足$a_n \leq a_{n+1} < b_{n+1} \leq b_n, b_n - a_n < 2^{-n}$, 那么对于每个$n$都存在一个可计算数$\alpha$，

$$a_n \leq a \leq b_n。$$

(A)

对于直觉主义而言，“规则”就是构造法。对于从0开始的n，2^{-n}等于二进制数1, 0.1, 0.01, 0.001,等等，所以这个规则的结果是一列相邻距离逐渐接近的有理数列——对于每一个新的n值，距离的值都小（右移）了一位。通常我们将这个序列称为收敛序列。

A proof of this may be given, valid by ordinary mathematical standards, but involving an application of the principle of excluded middle.

按一般意义上的数学标准，可以对此给出一个有效的证明，但要涉及排中律的应用。

问题是这个规则的结果包含了一些无法通过某种方法确认的东西，例如π的无穷展开式中符合特定规律的连续数字。图灵稍后会给出一个更有图灵特色（Turingesque）的例子。

On the other hand the following is false:

There is a rule whereby, given the rule of formation of the sequences a_n, b_n *in* (A) *we can obtain a* D.N. *for a machine to compute* α.

(B)

从另一方面而言，下面的说法是错的：

存在这样一个规则：如果给定(A)中可以形成序列a_n和b_n的那个规则，那么可以得到一个描述数，一台机器将可以使用这个描述数计算α。

(B)

注意，他认为这个说法是错的。

That (B) is false, at least if we adopt the convention that the decimals of numbers of the form $m/2^n$ shall always terminate with zeros, zeros, can be seen in this way.

至少如果我们采用这一约定，即形如$m/2^n$的数的小数形式以零结尾，则(B)是错误的，这一点可以从以下方式中看出：

形为$m/2^n$的数是有理数的子集，它们和图灵机有特定的联系，因为这样的数用二进制表示将只有有限个1。例如，有理数12345/65536用二进制表示是：0.0011 0000 0011 1001 0000…，因为65536是2^{16}，只有小数点后面的前16位二进制数可能是非0的（取决于分子），其余位都是0。任何m小于2^n的形如$m/2^n$的数字都以至多几个非零二进制数位开始，以无限个零永远继续下去。

图灵将会给出一个构造a_n和b_n的规则，但是这个规则基于的是另一台机器$\mathcal{N}$所打印出来的序列。

Let $\mathcal{N}$ be some machine, and define c_n as follows: $c_n = \frac{1}{2}$ if $\mathcal{N}$ has not printed a figure 0 by the time the n-th complete configuration is reached $c_n = \frac{1}{2} - 2^{-m-3}$ if 0 had first been printed at the m-th

[546]

complete configuration ($m \leq n$). Put $a_n = c_n - 2^{-n-2}$, $b_n = c_n + 2^{-n-2}$.

令 $\mathcal{N}$ 表示某个机器。定义c_n为：如果在第n个完全格局时 $\mathcal{N}$ 还没有打印出符号0，则$c_n = \frac{1}{2}$；如果在第m（$m \leq n$）个完全格局时第一次打印了0，则$c_n = \frac{1}{2} - 2^{-m-3}$。定义$a_n = c_n - 2^{-n-2}$, $b_n = c_n + 2^{-n-2}$。

让我们看看例子。假设 $\mathcal{N}$ 是一台打印序列1111101…的机器。下面是c_n、a_n和b_n的计算过程。

n	序列	c_n	a_n	b_n
0	1	$\frac{1}{2}$	$\frac{1}{2}-\frac{1}{4}=\frac{64}{256}$	$\frac{1}{2}+\frac{1}{4}=\frac{192}{256}$
1	1	$\frac{1}{2}$	$\frac{1}{2}-\frac{1}{8}=\frac{96}{256}$	$\frac{1}{2}+\frac{1}{8}=\frac{160}{256}$
2	1	$\frac{1}{2}$	$\frac{1}{2}-\frac{1}{16}=\frac{112}{256}$	$\frac{1}{2}+\frac{1}{16}=\frac{144}{256}$
3	1	$\frac{1}{2}$	$\frac{1}{2}-\frac{1}{32}=\frac{120}{256}$	$\frac{1}{2}+\frac{1}{32}=\frac{136}{256}$
4	1	$\frac{1}{2}$	$\frac{1}{2}-\frac{1}{64}=\frac{124}{256}$	$\frac{1}{2}+\frac{1}{64}=\frac{132}{256}$
5	0	$\frac{127}{256}$	$\frac{127}{256}-\frac{1}{128}=\frac{125}{256}$	$\frac{127}{256}+\frac{1}{128}=\frac{129}{256}$
6 …	1	$\frac{127}{256}$	$\frac{127}{256}-\frac{1}{256}=\frac{126}{256}$	$\frac{127}{256}+\frac{1}{256}=\frac{128}{256}$

在第一个0出现在序列里之前，c_n都是$\frac{1}{2}$。如果第一个0出现在序列中的位置是m（例子里是5），那么在n大于或等于m的情况下，c_n就等于$(2^{(m+2)}-1)/2^{(m+3)}$。

Then the inequalities of (A) are satisfied,

那么(A)中的不等式成立。

a_n的值总是在增加，b_n的值总是在减少，a_n和b_n差的绝对值总是小于2^{-n}。（实际上这个差总是2^{-n-1}。）

and the first figure of α is 0 if $\mathcal{N}$ ever prints 0 and is 1 otherwise.

如果 $\mathcal{N}$ 打印出0，那么α的第一个符号是0；如果 $\mathcal{N}$ 没有打印出0，那么α的第一个符号就是1。

在上面的例子里，极限显然是127/256，所以机器计算出来的α序列是011111110000…，表示那个有理数。如果在n等于4的时候序列出现了第一个0，那么极限是63/128，α序列就是011111100000…。仅当0从不在序列里出现的时候，极限才是$\frac{1}{2}$，这时候序列等于100000000000…。我们现在显然有了计算a_n和b_n的规则，但是这个规则建立在由其他机器 $\mathcal{N}$ 打印出的序列基础上，这样的规则需要有一个过程去判定像 $\mathcal{N}$ 这样的机器是否会打印出0。

如果 $\mathcal{N}$ 从未打印0，那么α的第一个符号是1；如果 $\mathcal{N}$ 打印过0，那么α的第一个符号是0。于是，

If (B) were true we should have a means of finding the first figure of α given the D.N. of $\mathcal{N}$: *i.e.* we should be able to determine whether $\mathcal{N}$ ever prints 0, contrary to the results of §8 of the paper quoted.

如果(B)是正确的，给定 $\mathcal{N}$ 的 D.N.，我们就有方法可以找到 α的第一个符号，即我们将能够判定 $\mathcal{N}$ 是否打印过0，这与所引用论文中§8的结果是矛盾的。

这里的“引用论文”指的是图灵的原始论文。接下来的一句话有点让人震惊，它乍看上去好像是错误的（就像我初见之时一样）：

Thus although (A) shows that there must be machines which compute the Euler constant (for example) we cannot at present describe any such machine, for we do not yet know whether the Euler constant is of the form $m/2^n$.

这样，虽然(A)表明肯定存在一些机器能够计算像欧拉常数这样的数，但我们目前还描述不出任何这样的机器，因为我们并不知道欧拉常数是否是$m/2^n$的形式。

图灵在这里提到的欧拉常数并不是作为自然对数基的那个e，而是不怎么著名的欧拉常数γ（gamma），它的计算式是：

$$\gamma = \lim_{n\to\infty}\left(1+\frac{1}{2}+\frac{1}{3}+\cdots\frac{1}{n}-\ln(n)\right)$$

或者换一种更有启发性的表达式，即是函数l/x的和式与积分的差：

$$\gamma = \lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i}-\int_{1}^{n}\frac{1}{x}\mathrm{d}x\right)$$

其近似结果是0.57721566490153286…。

这个欧拉常数也称为欧拉–马歇罗尼常数。欧拉在1734年提出计算这个常数的公式，并在1781年计算出了这个常数的前16位。马歇罗尼（1750—1800）之所以在这个常数上留了名，是因为他在1790年将它精确到了前32位（虽然只有前19位是正确的），并第一次用字母γ来表示这个常数。[27]

图灵并不是随意选择这个常数的，这个常数最著名的一个性质是，没有人知道它是有理数还是无理数。也没人知道，如果它是无理数，那么是代数数还是超限数。1937年人们不知道，直到2008年，人们依然不知道。

本章开头引用希尔伯特的话时提到了“难以接近”的问题。实际上，应该在那句话前加上：“考虑任何铁定未解的问题，如欧拉考马歇罗尼常数C是否是无理数，或者是否存在无穷多个形如2^n+1的素数。无论这些问题看似如何地难以接近……”欧拉—马歇罗尼常数被认为如此地难以接近，以至于希尔伯特并没有把它列入新世纪数学界要挑战的23个难题之中。

如果欧拉常数是有理数，那么它的分母可能是2的幂次方。如果这样，它的二进制表示将以无尽的0结尾。

如果用一台图灵机计算γ，显然没有通用过程可以判定γ是否形如$m/2^n$，因为这相当于判定这个机器是否能打印出有限个1，而这是不可能的，这很清楚。

图灵提出了一些更为严重的问题：必须知道欧拉常数是不是有理数，才能定义计算欧拉常数的图灵机，即图灵机自己必须“知道”γ是否以无尽的0结尾。这让那些真正动手编写算法计算γ的成千上万位数的人很惊奇。

不过，图灵机有一个关键点，它和图灵机不能消除已经打印的符号有关：当一个数字是$m/2^n$形式（m小于2^n）时，我们期望图灵机在前n位后都打印0。但一台计算欧拉常数的机器并不会这样做，因为计算欧拉常数的算法会用越来越小（但有限的）的项去逼近它。如果欧拉常数确实是$m/2^n$的形式，机器为了计算它的确切值就必须知道这点。否则，机器只会不停地逼近，因为它的逼近目标不是精确固定的。前n位之后的非0位就是错的，而且很成问题，因为这些位在图灵约定里是不能擦除的，但这些非0位又是不可避免的。

不管你如何欣赏图灵对欧拉常数问题的这番精彩的分析，都可能会发现他的解决方案比问题本身更糟糕。

> This disagreeable situation can be avoided by modifying the manner in which computable numbers are associated with computable sequences, the totality of computable numbers being left unaltered. It may be done in many ways* of which this is an example.

---

* This use of overlapping intervals for the definition of real

numbers is dud originally to Brouwer.

通过修改可计算数与可计算序列，或者与其余不被改变的可计算数的总和的联系，可以避免这种令人讨厌的情形。这可以用多种方法*实现，下面是一个例子。

---

* 这种使用重叠区间定义实数的方法最初源于布劳威尔。

如果笼罩在该节论文上的直觉主义气息在之前还不够明显的话，那么这里的脚注直指布劳威尔对实数的定义就是最好的证明。

实数的连续统一直以来都是个充满问题的概念，因为它同时涉及离散和连续两方面的性质。每一个实数在连续统上似乎都是精确的一点，但如果说连续统是由所有这些离散的点组成的，又会让人觉得很不舒服，特别是在康托尔告诉我们这些离散的点甚至都不可数以后。

布劳威尔尝试用一种既保留连续统的连续和离散性质又可以避免实无穷的方法来定义实数。

布劳威尔为这一过程发明的工具就是“选择序列”。选择序列有几种变形，但为了构造一个实数，它们都是有理数对的无穷序列，每个后续对定义了嵌套在前一间隔内的间隔。例如，下面是一个可能的选择序列，每个括号里表示一个嵌套的有理数对。

[3,4]
[3.1,3.2]
[3.14,3.15]
[3.141,3.142]
…

从经典意义上说，这个选择序列是在收敛的，我们激动地发现它似乎收敛于π。然而，在谈及布劳威尔的选择序列时，有必要避开“收敛”的概念，因为它暗含实无穷的意思。这个序列的每一个元素定义了两个端点间的连续区间。选择序列通过这样保持连续的性质。这个选择序列并没有极限π。相反，它保持着一个叫做halo[28]的类型，它在的的附近。当序列越来越长，区间越来越小的时候，这个halo会变得越来越小，但halo永远不会收缩到精确的无量纲无理数无π上。

一位布劳威尔学者如此解释布劳威尔的选择序列和传统极限的不同：

有必要强调，选择序列直观上是随着时间而增长的，它本身就

> 是实数……在布劳威尔的解释里，人们能够很好地理解实数，因为它就是这个不断随时间增长的序列本身。选择序列并不是用来逼近超真实里的实数的方法。[29]

在直觉主义的连续统里，实数总是不完全的——没有终点，永远不会终结，并展现出一个非0的维度。

上面展示的选择序列可以由一些算法生成，因此它也叫做“似定律”（lawlike）选择序列。也有一些“非似定律”选择序列，其中的每个元素是由一些决定序列如何发展的代理（如数学家）选择的。数学家甚至通过抛硬币来决定选择序列中的元素。

> 我们可以将非似定律选择序列看成一种直觉，即使它与似定律的直觉在某些方面相当不同。非似定律选择序列依然是一个主体（subject）或超越自我（transcendent ego）随时间执行的序列，其中只有一部分是确实完成的。事实上，我们只完成了它的有限的初始段，非似定律选择序列就像朝着实数的方向不停地勾勒地平线（但地平线的那边还有地平线）。我们应该把它看成一种“自由转换的手段”。应该指出，我们这里说的选择序列是一种过程，一种随时间演化的行动序列。[30]

选择序列也可能是似定律和非似定律选择序列的组合。例如，在多个非似定律选择序列上执行事先确定的算法操作。

如果两个选择序列从相同的元素开始，并持续保持一段时间的相等，那么它们可以视为相等。例如：

$$
\begin{array}{cc}
[3,4] & [3,4] \\
[3.1,3.2] & [3.1,3.2] \\
[3.14,3.15] & [3.14,3.15] \\
[3.141,3.142] & [3.141,3.142]
\end{array}
$$

过了一段时间，这个序列可能变得不相等，而是区间重叠了：

$$[3.141, 3.14175] \quad [3.14125, 3.142]$$

或许再过一段时间，序列又相等了：

$$[3.14125, 3.14175] \quad [3.14125, 3.14175]$$

接下来会发生什么，我们只能去猜。

这样看来，似乎似定律选择序列可以由产生它的算法完全定义，因此它能表示连续统上的离散点。

> 即使这样，对于选择序列表示直觉主义连续统上一点的正确理解应该是，这个选择序列必须看作是进行中的选择序列，不管它是似定律的或非似定律的。对于非似定律选择序列，这点比较容易理解；对于似定律序列，这点也应该相同。否则，如果把似定律选择序列看成完成的对象，那么我们又会被引入用经典的原子论观点看待直觉主义连续统上的点的圈套里，这样连续统就会中断。换句话讲，直觉主义连续统上的点永远不表示“是什么”，而表示“变为什么”，连续统通过这个性质而保持。[31]

读过图灵论文的读者应该感到上面的这些概念有点熟悉，因为我们了解了图灵机。考虑一个能表示实数的图灵机（或者它的描述数），但是这台图灵机总是经过一段时间才打出一些数位符号。它永远不会终结，我们永远不可能通过它的描述数判定这个实数将变成什么样。二进制的读π/4是：

110100100001111110110101010001000…

（我之所以用（π/4而不是而/是因为为π/4在0和1之间。）一台计算之π/4中各个数位的图灵机可以看成是在计算它的选择序列。当机器计算到第一位1时，这一位并不表示数字0.1，它表示的是从0.10000000…到0.11111111…的区间，因为这是开头为0.1的实数可能所在的范围。记住，序列0.11111111…等于1，于是这台机器计算产生的嵌套的选择序列是：

[1.1,1.0]
[0.11,1.00]
[0.110,0.111]
[0.1100,0.1101]
[0.11001,0.11010]
[0.110010,0.110011]
[0.1100100,0.1100110]
…

在这个意义上，一台普通的遵守图灵约定（也就是不会擦除已经打印的符号）的图灵机，会产生一个表示可计算实数的布劳威尔选择序列。这个过程一直持续，不会终结。

如果图灵在世，用我的方法来看待上述这种巧妙的联系，他是不会

认可的。诚然，他没有我今天的条件，能够看到21世纪学者对布劳威尔那些晦涩文章所作的解读。如果那时图灵读过布劳威尔的文章，或者通过间接渠道获得布劳威尔的思想，那么这篇文章可能会变得与布劳威尔1928年的论文《连续统的结构》（*Die Struktur des Kontinuums*）相似[32]。布劳威尔的论文讲的是将选择序列作为一种树形结构，通过重叠区域去覆盖连续统中的片段。这可能会让图灵想到，如何将可计算序列转换为可计算数。另外，为了解决欧拉常数的问题，图灵还可能考虑将其机器的计算范围扩展到0和1区间以外，因为他的机器那时只能计算0和1区间内的数。

在图灵原始的想法里，只要加一个二进制小数点，一个可计算序列就变成了实数。不过，修订后的论文里的表述更为复杂一些。

> Suppose that the first figure of a computable sequence γ is i and that this is followed by 1 repeated n times, then by 0 and finally by the sequence whose r-th figure is c_r; then the sequence γ is to correspond to the real number
>
> $$(2i-1)n+\sum_{r=1}^{\infty}(2c_r-1)\left(\tfrac{2}{3}\right)^r.$$

> 假设可计算序列γ的第一个数是i，它之后跟着n个连续的1，然后是0，最后是一个第r个数是c_r的序列，那么这个序列γ对应的实数就是：
>
> $$(2i-1)n+\sum_{r=1}^{\infty}(2c_r-1)\left(\frac{2}{3}\right)^r$$

符号i表示数的正负号：0表示负号，1表示正号。n个连续的1表示数的整数部分。0起到分隔作用，它的功能就像一个二进制小数点。接下来的那个序列表示数的小数部分，它的形式总是为：

$$\pm\frac{2}{3}\pm\frac{4}{9}\pm\frac{8}{27}\pm\frac{16}{81}\pm\frac{32}{243}\pm\cdots$$

其中c_r表示每一项是正号（等于1时）还是负号（等于0时）。

例如，一台机器打印出下面的序列：

1 111110 11011…

我在数字间留了空隙，是为了便于转换。它的第一位是1，表示是个正数。接下来的5位是连续的1，以0结尾，所以正数部分是5。小数部分是：

$$\frac{2}{3}+\frac{4}{9}-\frac{8}{27}+\frac{16}{81}+\frac{32}{243}$$

在这一步后已经完成的数字是 $5\frac{278}{243}$ 或 $6\frac{35}{243}$ 。注意，被计算的这个数字的整数部分是6而不是5。如果表示小数的那些数位全是1，那么小数部分就是：

$$\frac{2}{3}+\frac{4}{9}-\frac{8}{27}+\frac{16}{81}+\frac{32}{243}+\cdots$$

它收敛于2。类似地，如果表示小数的那些位全是0，那么小数部分收敛于，2。一个整数部分被编码为*N*的序列其实可以还原为*N*-2到*N*+2，相当于创造了一个重叠区间。

If the mechine which computes γ is regarded as computing also this real number then （B) holds.

如果计算γ的机器也可以看成是计算这个实数，那么(B)成立。

也就是说，如果存在一个规则可以公式化表达接近某个数的序列a_n和，b_n我们就可以得到这台机器的描述数。机器在内部计算a_n和b_n，然后根据是减去或加上$(2/3)^n$才能使计算出的数在新的区间范围里来选择打印0或者1。

在这个过程中，我们只能通过单一的、基于收敛边界的方法来计算数。我们再也没法用一台简单的机器来计算像$\frac{1}{2}$或$\frac{1}{4}$这样的有理数。这些数必须像其他数那样通过逐渐逼近来计算。例如，$\frac{1}{2}$的序列现在

是：

001010 1101 0100 1101…

前两位表示正号和二进制小数点。后面的几位表示2/3, -/4/9, 8/27, 416/81，等等。这里给出的这部分序列用大于3位十进制或者10位二进制的精度来表示$\frac{1}{2}$。

这种做法的优点是，机器不需要知道它正在计算的是一个有着$m/2^n$形式的有理数。它不需要知道该什么时候放弃计算而在末尾加无穷的0。即使一个以无穷的0或1结尾的序列也要与一个表示为无限个递减但有有限个项的数相对应。

还有另一些方法计算$\frac{1}{2}$。下面是其中一种：

0 10 0101 0010 1011 0010…

第一位表示正号，接下来的两位表示数字1。再接下来的几位表示再-2/3, 4/9, -/8/27, 16/81等项，刚好与前面第一个序列相反。

> The uniqueness of representation of real numbers by sequences of figures is now lost, but this is of little theoretical importance, since the D.N.'s are not unique in any case
>
> The Graduate College,
>
> Princeton, N.J., U.S.A.

> 通过数字序列来表示实数的方法现在并不唯一了，但从理论上讲这并不重要，因为任何情况下描述数都不是唯一的。
>
> 研究生院
>
> 普林斯顿，美国新泽西州

这就是图灵这篇短文的全部内容。图灵这次对假设和推理的改变给我们带来的那点迷惑似乎并不遗憾。现在，我们有了一个对可计算数的不怎么繁琐的数学定义，这或许会让我们稍感满足，但是依然很难弥补我们从哲学角度对连续统问题茫然无措的迷失感。

对连续统的设想

---

[1] 参见Hal Hellman，*Great Feuds in Mathematics: Ten of the Liveliest Disputes Ever*（Wiley，2006），它给出了详尽的介绍。

[2] 莫里斯·克莱因，Mathematics: The Loss of Certainty（Oxford University Press，1980），323。
[3] 罗杰·彭罗斯，The Road to Reality: A Complete Guide to the Laws of the Universe（Alfred A. Knopf，2005），13。
[4] 引自Ben H. Yandell，The Honors Class：Hilbert’s Problems and Their Solvers（A. K. Peters，2002），395。
[5] 亚里士多德，Physics，译者Robin Waterfield（Oxford World’s Classics，1996），Book III，第4章，65。
[6] 同上，Book III，第6章，72。
[7] 同上，Book III，第6章，73。
[8] 同上，Book III，第8章，76-7。
[9] 斯蒂芬·克尔纳，“Continuity”in Paul Edwards，ed.，*The Encyclopedia of Philosophy*（Macmillan，1967），Vol. 2，205。
[10] 这个引用并没有在克罗内克的作品中找到。它第一次出现在1893年的“Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk”。参见William Ewald，ed.，*From Kant to Hilbert*：*A Source Book in the Foundations of Mathematics*（Oxford University Press，1996），Vol. II，942。在1922年的“The New Grounding of Mathematics。 First Report”中， 希尔伯特用单数形式引用它。原话为“Die ganze Zahl schuf der liebe Gott, alles andere ist Menschenwerk”参见*From Kant to Hilbert*，Vol. II，1120。
[11] Harold Edwards，“Kronecker’s Place in History”，in William Aspray and Philip Kitcher，eds.，*History and Philosophy of Modern Mathematics*（University of Minnesota Press，1988），139-144。
[12] 格奥尔格·康托尔，“Foundations of a General Theory of Manifolds：A Mathematico-Philosophical Investigation into the Theory of the Infinite”，in *From Kant to Hilbert*，Vol. II， pgs.895-896。
[13] 沃尔特，“Brower’s Intuitionist Program”，in Paolo Mancosu，ed.，*From Brouwer to Hilbert: The Debate on the Foundations of Mathematics in the 1920s*（Oxford University Press，1998），5。
[14] 布劳威尔，“Intuitionism and Formalism”，*Bulletin of the American Mathematical Society*，Vol. 20（1913），81-96。
[15] 同上。
[16] 布劳威尔，“On the Significance of the Principle of Excluded Middle in Mathematics, Especially in Function Theory”（1923），in jean van Heijenoort. ed.，*From Frege to Gödel*：*A Source Book in Mathematical Logic*，1879-1931（Harvard University Press，1967），337。

[17] 康斯坦斯·雷德，*Hilbert*（Springer，1970，1996），184。
[18] Jonathan Borwein，The Brouwer-Heyting Sequence，http://www.cecm.sfu.ca/~jborwein/brouwer.html。
[19] F1oy E. Andrews，“The Principle of Excluded Middle Then and Now: Aristotle and *Principia Mathematica*”，*Animus*，Vol. 1（1996），http://www2.swgc.mun.ca/animus/1996vol1/andrews.pdf。
[20] 希尔伯特，“On the Infinite”（1925），参见*From Frege to Gödel*一书，375-376。
[21] G. Kreisel 和纽曼，“Luitzen Egbertus Jan Brouwer.1881-1966”，*Biographical Memoirs of Fellows of the Royal Society*，Vol. 15（Nov. 1969），39-68。
[22] 例如，参见Oliver Aberth，*Computable Analysis*（McGraw-Hill，1980）。
[23] 在*From Brouwer to Hilbert*一书中重印，28-35。
[24] William Aspray，The Princeton Mathematics Community in the 1930s: An Oral-History Project。An interview with Alonoz Church at the University of California On 17 May 1984，http: //www.princeton. edu/ ~mudd/finding_aids/mathoral/pmco5.htm。
[25] 邱奇，“A Set of Postulates for the Foundation of Logic”，*The Annals of Mathematics*，second Series，Vol. 33，No. 2（Apr. 1932），346。
[26] 克莱尼，“Origins of Recursive Function Theory，” *Annals of the History of Computing*, Vol. 3. No. 1（Jan. 1981），62.
[27] Julian Havil，*Gamma*：*Exploring Euler’s Constant*（Princeton University Press，2003），89。
[28] 这一术语来自Edmund Husserl（1859——1938），参见 Mark van Atten，Dirk van Dalen，and Richard Tieszen，“Brouwer and Weyl: The Phenomenology and Mathematics of the Intuitive Continuum”，Philosophia Mathematica，Vol. 10，No. 2（2002），207。
[29] Mark van Atten，*On Brouwer*（Wadsworth，2004），31。
[30] van Atten，van Dalen，and Tieszen，“Brouwer and Weyl: The Phenomenology and Mathematics of the Intuitive Continuum，”212。
[31] van Atten，van Dalen，and Tieszen，“Brouwer and Weyl: The Phenomenology and Mathematics of the Intuitive Continuum，”212-213。
[32] 布劳威尔译为“The Structure of the Continuum”，*From Kant to Hilbert*，Vol. II，1186-1197。

第四部分
题 外 话

第17章 万物皆是图灵机？

不管你对图灵的概念或原理有多深的了解，都不会对你构建一台真正的计算机有所帮助。数字计算机是由半导体管和其他一些如继电器和真空电子管等开关机制部件构建的。这些半导体管组装成逻辑门，从而实现简单的逻辑功能。寄存器和累加器等高层次的组件都是由这些逻辑门构成的。[1]

图灵机是由什么构成的呢？图灵从来没有告诉过我们。图灵并没打算让他的机器成为实际计算机的蓝图。图灵机是对计算的一种简单抽象的模型，这种计算既可以由人完成，也可以由机器完成。图灵创造图灵机的初衷是为了一个特定的目的：证明对于一阶逻辑并不存在通用判定过程。只是后来，我们发现这种想象中的机器对理解计算理论也有重大的辅助作用。这个转变花了20年的时间，也就是在图灵机成为了我们目前称之为“计算机科学”这一学科的研究对象之后。

将图灵机应用于其他目的（而不是单纯为了证明判定性问题），需要在某种程度上重新改造图灵机。大多数图灵机会永不停息地计算0到1之间某个实数的数位。数学里的常见任务，也是计算机编程里的常见任务，就是函数的计算。一个函数需要一个或者几个数作为输入——称作函数的参数。基于输入，函数会输出计算的结果——称作函数的值。

一类重要的函数就是数论函数，之所以这么称呼是因为它的输入和输出仅限于自然数。图灵在其论文的第10节（本书第219页）中发明了一种技巧来计算数论函数，即打印由单个0间隔的连续1的串。第一个0前的连续1的个数表示参数为0的函数值，第二个0前（第一个0后）的连续1的个数表示参数为1的函数值，等等。

对图灵的数论函数持怀疑态度的一位数学家是斯蒂芬·科尔·克莱尼。克莱尼是邱奇在普林斯顿的学生，他在1934年取得博士学位，之后开始在威斯康辛麦迪逊大学教书。

克莱尼后来写道：“虽然我很向往图灵所构想的机器的非凡能力，但我依然对他用如此简单的方法将此应用到数论函数的计算中表示怀疑。在任何情况下，只有完全函数$ø(x)$才能用这种方法计算。”[2]图灵

的方法对部分函数不适用，这些函数只对自然数的部分子集成立。

1941年开春，克莱尼在威斯康辛麦迪逊大学教授的一个数学基础研讨班上开始寻求一种不同的解决方法。克莱尼重新改进的图灵机在他1952年出版的经典图书《元数学引论》的第8节中占据了重要的位置。

克莱尼版本的图灵机仍然是读符号，写符号，沿纸带左右移动。不过，它只限于一种符号，即一个简单的竖线，称为tick或tally符号。机器只有这种符号和空格。自然数以由空格分割的一系列连续格上的tick符号表示。克莱尼的自然数以0开始，一个tick符号表示0，2个tick符号表示1，以此类推。克莱尼好像是第一个在文章中将图灵机纸带的例子作为插图的人。[3]

图灵机通常以一个空白纸带开始。而克莱尼改进后的机器则从一个已经编码了一个或几个隔着空格的连续tick串符号（作为函数输入）的纸带开始。克莱尼的机器稍后计算函数的值，并将数字编码到纸带上。克莱尼给出的第一个例子是计算后继函数（即计算被编码的数的下一个数）的值。它只是简单地将另外的tick符号打印在现有tick符号串的后面，因此很方便。

克莱尼的函数计算机器只需要在一段有限的时间内进行计算，当机器完成计算的时候就停止了。这种机器并没有特殊的“停止”或“停机”格局，但是有克莱尼所谓的“被动状态”，即已经不存在机器可以到达的位置。当机器被指令转移到不存在的格局时，“这个机器被称为停止了，我们称它停止时候的状态为终止状态或者输出”。[4]

在图灵的概念里，一台好的机器——图灵称之为非循环机，即符合要求的机器，是永不停止的。经过克莱尼改造后，一台好的机器将在计算完函数后停止运行。一台陷入了无限循环而无法停止的克莱尼机是“不好”的机器。显然，克莱尼的机器更接近传统的数学观念，即函数接受输入并经过有限步骤输出结果。

就如第15章讨论的，到了1936年，已经存在3种形式的计算有效性直观表示，它们是：

〉 图灵机；

〉 1934年，哥德尔基于雅克·赫尔布兰德的建议而定义的递归函数，克莱尼做了进一步的发展；

〉 邱奇及其学生（主要是克莱尼）发展的λ可定义函数。

这三种不同形式表示方法的等价性，一部分是由图灵在1936年关于可计算数的论文的附录中建立起来的，更严格的说明是在其1937年的论文“可计算性和λ可定义性”中。另外，斯蒂芬·克莱尼在1936年的论文“λ

可定义性与递归性”中也有所说明。现在“递归函数“与”可计算函数”几乎表达同一意思。

斯蒂芬·克莱尼是第一个提出这些形式化表示方法如何直观表达可计算性的人。他在《元数学导论》一书中，第一次明确提出邱奇论题：“每一个有效可计算函数（或者有效可判定谓词）都是一般递归的。”克莱尼又在其后的两章中说：“图灵论题，即每一个被自然认为可计算的函数在他的定义下（即通过一台图灵机）也是可计算的，它实际上与邱奇论题等价……”[5]

在其1967年出版的书中，克莱尼将两个论题结合在了一起：

> 图灵论题和邱奇论题是等价的。我们应该将它们统一称为“邱奇论题”，或者“邱奇邱图灵论题”，以表明它和三种形式化表示方法之一的“图灵机”有关。[6]

自那以后，”邱奇邱图灵论题”成为了最适当的术语。

《元数学导论》显然是一本面向数学家们的书。6年以后，另一本经典著作帮助我们跳出纯数学的视野，从计算机科学的角度阐述问题。

马丁·戴维斯于1928年生于纽约市。他在1950年获得了普林斯顿大学的博士，其博士论文是《递归的不可解性理论》。戴维斯的论文导师是邱奇，也是克莱尼（1934年）和图灵（1938年）的导师。

在伊利诺伊大学教书时，戴维斯开始称判定图灵机能否完成计算的问题为“停机问题”，也许最早是在1952年。[7]这个术语在1958年戴维斯出版了《可计算性和不可解性》一书后而广为人知。在这本书的前言里，戴维斯诡秘地写道：“虽然这一卷里真正新的东西很少，但是我对相应主题的安排和论述可能会让专家感到新颖，”随后他做了说明，“特别是，图灵机的概念是这本书行文论述的关键。”[8]

在克莱尼的《元数学导论》中，图灵机直到第321页才出现，在第13章之前也没有更深的提及；而在戴维斯的《可计算性和不可解性》中，图灵机在第1章的第一页就出现了。

和克莱尼一样，戴维斯把自然数表示为连续的tick符号，并用图灵机来计算函数。用图灵机计算加法、减法、乘法的例子出现在书的第12页。

虽然《可计算性和不可解性》表面上是数学图书，但是戴维斯意识到这本书“由于和某些哲学问题以及数字计算机理论相关，一些非数学家也可能有兴趣一读”。[9]

为进一步强调不同，《可计算性和不可解性》作为McGraw-Hill出版公司”信息处理和计算机系列丛书“中的一部作品出版。即使在这套丛

书中，这本书也是独一无二的。其他书都关注于计算机硬件和程序设计的“实用”主题。在1958年和1959年，该丛书出版了《模拟仿真：场问题的求解》、《高速率数据处理》、《数字计算机入门》、《数字计算机系统》和《数字计算机编程入门》。

马丁·戴维斯的《可计算性和不可解性》真正开创了将可计算性作为一门学科主题的研究，它后来成为计算机科学专业学生必修的一门课程。

在《可计算性和不可解性》的第70页，马丁·戴维斯引入了一个和图灵机密切相关的术语：

> 现在，令Z表示一个简单图灵机。关于Z，我们有如下判定问题：
>
> 对于一个给定的瞬间描述α，判定是否存在一个以α开始的对Z的计算。
>
> 也就是说，我们希望确定如果给定初始状态，那么Z会不会最终停止？我们将这个问题称为Z的停机问题[10]。

在第70页的最后，戴维斯构想了一个定理：“存在一种图灵机，其停机问题是递归无解的。”

戴维斯的书影响深远，一谈到停机问题就会联系到图灵机，尽管图灵原来设想的图灵机是永不停止的。

除了速度、存储能力、人机交互设备不尽相同外，现代计算机大体上类似。每台能模拟图灵机的计算机（这是最简单的需求）都是一台通用计算机。此外，一台通用计算机能够模拟任何其他通用计算机。

最初的一些很简单的计算机甚至还达不到图灵机的能力。显然，第一台至少能称为潜在通用计算机的是Z3，由康拉德·楚泽在1938年至1941年建造。[11]如果说是建造成功，第一台通用计算机应该算是查尔斯·巴贝奇在19世纪30年代建造的分析引擎，虽然它是由一堆齿轮而不是转换开关构成的。实际上，所有1944年以后生产的计算机都是通用计算机。

通用计算机的一个关键特性是可编程性。必须存在某个将指令集引入计算机的方法，使计算机能响应这个指令集。现代计算机将这样的指令集作为字节存储在内存里，称为机器码。在楚泽的机器里，指令集的编码用35mm的电影胶带上的打孔来表示。巴贝奇的机器则使用打孔的卡片，与控制纺织机的卡片有点像。

一些早期的计算机只能用很不灵活的指令序列进行编程。一个通用计算机必须能够根据上几步的计算结果跳过一些指令序列。这个特性我

们现在称为条件分支，这对于实现条件循环是必要的。

如果一门计算机编程语言能够模拟图灵机，它就经常被称为是图灵完备的。

最初在互联网上广泛使用的超文本标记语言（HTML）并不是用来为计算服务的，因此它显然不是图灵完备的。经常在HTML中使用的JavaScript就是图灵完备的。几乎所有现今使用的编程语言都是图灵完备的。任何图灵完备的编程语言都可以模拟任何其他图灵完备的编程语言。

图灵机不仅说明了进行有效计算的最基本需求，也说明了它的限制：没有任何一台现代计算机或一门编程语言能够比图灵机更强大，没有任何计算机或编程语言能够解决停机问题，没有任何计算机或编程语言能够判定其他计算机程序的未来运行状态。你不能使用更“先进”的编程语言或者不一样的机器来应付这种限制。你能做的，只能是加快计算机的工作速度。你可以拿出上千个处理器组成并行计算机集群，以进行大规模的并行计算，但是你不可能将无限带往我们生活的这个无助的有限世界里，哪怕一点点。

一些数学家不顾图灵机的限制，毅然钻入超计算领域，以试图让机器摆脱图灵的限制。图灵自己也部分推动了这项工作。他在自己1939年发表的晦涩的博士论文《基于序数的逻辑系统》（*Systems of Logic Based on Ordinals*）中称之为“神谕”，并写道：

> 假设我们拥有一种可以解决（不可判定）数论问题的不确定方法，那么可以称这种方法为神谕。对于这个神谕，我们除了知道它肯定不是一台机器外无法知道更多。在这个神谕的帮助下，我们可以构造一种全新的机器（称为o-机器），这个机器的某一个基本进程可以用来解决给定的数论问题。[12]

也许我们每个人都希望生活中有这样的神谕，来帮助我们解决棘手的问题。探索超计算的研究学者基于神谕的概念，将其他特性引入图灵机以使其不再受到先前的限制。虽然出现了一些很有趣的数学构造，但是这样的超计算机并不切合实际，因为它违反了一些基本的物理定律，比如加速时间使得每步计算都是前一步计算的1/2时长。马丁·戴维斯形容超计算是谜一样的问题，并将它与三等分给定角和发明永动机这样的问题作比较。[13]

在我看来，超计算的研究对于解答计算的普遍性是很有价值的。图灵设计出假想的机器，以刻画人类计算者在执行特定算法时进行的基本操作。他发现，图灵机具有一些固有的限制。那以后的几十年，我们建

造了和图灵机等价的计算机，因此同样面临这些限制。我们还没有找到有效的方法来摆脱这些限制。

基于上述原因，计算的普遍性（在能力和限制上）对于任何数据处理活动而言似乎都是基本存在的。这些限制就像是热力学定律一样是自然界内在的规则。

如果图灵机的内在限制不能在遵守物理定律的前提下被超越，那么对于那些执行计算或逻辑运算的内在机制而言，这又暗示着什么呢？当我们从探索人的思维和宇宙自身的角度来考虑这两个最重要的（也许甚至有些令人烦恼的）“内在机制”时，这个问题变得最为深刻。

严格地说，图灵定理只涉及图灵机和机械算法的等价问题，并不一定意味着不存在超越图灵机计算能力的计算机器，这样的机器也未必一定违反某个物理定律。[14]

也许我们漏掉了什么，也许存在某种神奇的物理机制能够执行非常强大的计算操作，而这种机制无法在图灵机上模拟出来。图灵机真的有助于我们对人类思维和宇宙的理解吗？或者我们只是愚蠢地把一个非常复杂的问题简单化到图灵机的层次上？

图灵机在数学和计算领域外的遗留问题出现在图灵1936年论文发表之后的几年，源于沃伦·麦卡洛克（1898—1969）和沃特·匹茨（1923—1969）的一次偶遇。

在底特律，青年沃特·匹茨聪明好学，自学了拉丁语和希腊语、哲学和数学，家里人都认为他是一个怪才。15岁的时候，他跑到了芝加哥。因为无家可归，沃特·匹茨大多数时间游荡在公园里，在那里，他遇见了一位名叫伯特的老人，他们在哲学和数学上有着相同的兴趣。伯特建议他读一读由芝加哥大学教授鲁道夫·卡尔纳普（1891—1970）写的一本出版于1937年的书，可能是《语言的逻辑句法》。沃特·匹茨读了这本书后，就径直前往卡尔纳普的办公室，与他讨论自己在书中发现的几个问题。这个叫伯特的老人就是鼎鼎大名的伯特兰·罗素。[15]

如果你不相信这个故事，那么下面的故事似乎更可信。伯特兰·罗素在芝加哥大学教书的时候，有一次散步从杰克逊花园经过，发现有个年轻人正在看卡尔纳普的书。于是，罗素和年轻人开始交谈起来，并把匹茨带到了卡尔纳普的办公室。[16]

还有一则故事。当匹茨12岁、还生活在底特律的时候，有一次被几个流氓追赶，躲进了图书馆。图书馆关门后，匹茨被困在了里面。他决定读怀特海和罗素的《数学原理》作消遣。这么一读就是3天，然后他给罗素写了一封信，指出了书里的一些错误。当罗素写信邀请他去剑桥时，匹茨决定成为一个数学家。[17]

有一点是可以确定的，匹茨在1938年上过罗素在芝加哥大学教的

课，同年他也到过卡尔纳普的办公室。卡尔纳普对这个年轻人印象深刻，想给他一份学生助教的工作，但是他并不知道匹茨的名字，所以无从找到他。[18]

匹茨是个“害羞、内向的孩子，他戴着眼镜，牙齿不齐，有拨弄头发的习惯，有轻微的神经颤抖症，走路时还经常会碰到东西”。[19]（在这以后的第二次世界大战期间，匹茨被征兵局划入4F级别[20]，并被认为处于“精神病发作前期”。但是匹茨后来参加了曼哈顿计划，并获得接触最高机密的许可。）[21]在匹茨访问了卡尔纳普的办公室几乎一年后，卡尔纳普终于找到了匹茨。匹茨开始跟着他学习逻辑学，同时在芝加哥大学上课，其中包括了红胡子的乌克兰人尼古拉斯·拉谢甫斯基（1899—1972）主持的研讨班。

拉谢甫斯基在基辅大学获得了理论物理的博士学位，1924年移民到美国。他对应用数学模型解决生物生长发育过程中的问题很有兴趣。在这个领域中，之前其他科学家都仅仅依赖于经验性的研究，并没有相关的科学实验方法。到了1934年，拉谢甫斯基提出了一个名字来形容他所研究的工作：“数学生物物理学”。1935年，他成为芝加哥大学第一名数学生物物理助教。1938年，一本名为《数学生物物理学》的书出版了，随后在1939年，诞生了名为《数学生物物理学报》的期刊，主要发表拉谢甫斯基及其理论追随者的一系列论文。[22]

1942年和1943年期间，匹茨接连在《数学生物物理学报》上发表了3篇论文。沃伦·麦卡洛克就是在这时听说了匹茨的这些研究工作。

沃伦·麦卡洛克在新泽西长大，一开始在哈弗福德学院，这是宾夕法尼亚州的一座“教友派”学校。当麦卡洛克1917年进入大学不久，哲学家鲁弗·琼斯（1863—1948）（他在这期间帮助创建了美国朋友服务委员会）就问了麦卡洛克一个问题：“你将会成为什么样的人？……你将会做什么事?”麦卡洛克说不知道，但是他说：“我想回答的是另一个问题：人可以认知的数是什么？可以认知数的人又是什么？”鲁弗·琼斯对此只能回应道：“朋友，看来你的人生要在忙碌中度过了。”[23]

后来，麦卡洛克去了耶鲁大学学习哲学和心理学，于1927年在纽约的内外科医学院获得硕士学位，并在贝尔维尤医院从事治疗严重脑损伤病人的工作。后来，他又去了洛克兰德州立医院从事治疗精神病患者的工作。[24]1934年，麦卡洛克回到了耶鲁。他的同事里有杜赛尔·德·巴瑞内（1885—1940），巴瑞内是使用化学成分马钱子碱作用在猫的大脑并观察猫的反应，从而探明大脑各部分对应功能这一实验方法的先驱。1941年，麦卡洛克搬到了伊利诺伊州，在伊利诺伊神经精神病学院工作。

麦卡洛克是个传奇人物。“有点像摩西：有一撮长长的胡子，浓密

的眉毛，眼睛发出奇异的光芒。很多时候，他看上去就像个疯子。他的眼睛是灰色的，当它们变得明亮而闪烁的时候，就像是一副眼镜。”[25]麦卡洛克很喜欢社交，“每天晚上都要喝一瓶苏格兰威士忌作为他和别人闲聊时的情绪催化剂”，[26]他也是个很会讲故事的人。（有关沃特·匹茨和伯特兰·罗素在杰克逊公园相遇的故事最早来源于麦卡洛克。）麦卡洛克会写诗，还经常论述他的哲学观点，到处炫耀自己的博学多才。

沃伦·麦卡洛克和沃特·匹茨事最初是怎么相识的，至今也不清楚。但是他们很快就情投意合，匹茨甚至搬来和麦卡洛克一起住。麦卡洛克一直想尝试建立一套大脑工作机理的形式化理论，而匹茨对于数学逻辑的通晓正是麦卡洛克所需要的。他们在麦卡洛克厨房的桌子上写出了名为《神经活动中内在意识的逻辑演算》的论文，1943年发表在拉谢甫斯基的《数学生物物理学报》上。麦卡洛克的女儿塔菲为他们第一次合作的这篇论文画了图解。[27]

得益于19世纪下半叶的研究，科学家们已经知道神经系统是由叫做神经元的细胞构成的，这些神经元似乎像网络一样连接在一起。20世纪的进一步研究指出，这些神经元就像开关一样，当刺激达到一定阈值时开关就被触发。[28]

对于麦卡洛克和匹茨来说，这些神经元就像逻辑开关，于是他们用卡尔纳普的标注方法对神经元进行命题逻辑意义上的建模。一个重要的、传统逻辑中并不存在的要素是，神经元逻辑存在输入与输出的延时特性。基于这样的延时，神经元就被组成为环状结构，从而信号可以在网络内保持一定时间的有效期。麦卡洛克和匹茨的论文为这个模型定义了几个公理，并进而证明了几个定理。

《神经活动中内在意识的逻辑演算》并没有参考前人太多的成果。这篇论文注明的参考文献只有卡尔纳普的《语言的逻辑句法》、希尔伯特和阿克曼的《数理逻辑原理》，以及怀特海和罗素的《数学原理》。在这篇19页论文的第15页，麦卡洛克和匹茨在做总结的时候透露了一些他们所参考的更广泛文献来源：

> 首先，如果每个网络接有纸带，它的扫描头与传入神经连接，并且有适合的传出神经进行必须的操作，那么它就只能计算图灵机能够计算的那些数；其次，图灵机能够计算的数也能够被这样的网络所计算……这就从心理学的角度，论证了图灵关于可计算性的定义，以及与其等价的邱奇的λ可定义、克莱尼的一般递归：如果任何一个数能够被有机体计算出来，那么它也可以被这些等价定义计算出来，反之亦然。[29]

几年后的1948年，麦卡洛克将大脑与图灵机的联系表述得更加清楚。他解释说，他正努力寻找一种方法来发展神经生理学理论。

> 直到我看了图灵的论文，才发现找到了正确的途径，也感谢匹茨那些必要的逻辑演算的帮助。我们认为，我们正在做的（我想我们获得了相当的成功）是将大脑视为一台图灵机；除非大脑出错，机能失常，否则大脑所进行的就是这台图灵机进行的功能……令人高兴的是，一些非常简单的假设就足以说明神经系统可以计算任何可计算的数。大脑就是一种仪器，一台图灵机（如果你喜欢这个词）。[30]

一直到后来（1955年），麦卡洛克依然坚持他的这个观点："匹茨和我已经说明大脑就是图灵机。任何图灵机都可以由神经元构成。"[31]虽然现在的科学技术还不足以将这种等价性转换为实际用途。

> 一个理论上的问题是，你能设计出一台机器做大脑所能做的事情吗？答案是：如果你能用有限的而且不含糊的方法说明大脑能够做的事情，我们就可以设计出这样一台图灵机。匹茨和我证明了如何完成这样的构造。但是你能说出大脑能够做的事情吗？[32]

要不是引起了两位20世纪计算领域的巨匠诺伯特·维纳和冯·诺依曼的注意，麦卡洛克和匹茨的论文可能会因无人知晓数学生物物理学而失去光彩。

继约翰·斯图亚特·穆勒后，诺伯特·维纳是又一个被那种臭名昭著的天才家庭教育模式培养长大的例子。这两个人后来都曾在回忆录中写过小时候被满脸胡须的父亲塑造为天才儿童的体验。而对于维纳，小时候留下的伤疤终其一生都难以愈合。多年来，他要和未经诊断的抑郁症抗争，与其天才般的研究热情相伴的是难以名状的暴躁情绪和产生自杀倾向的绝望状态。

11岁时，维纳就进入了塔夫斯大学，14岁时拿到了数学学士学位，18岁成为哈佛大学历史上最年轻的博士毕业生。维纳的父亲一直对新闻界说，他的儿子"并不是神童"，事实上"很懒惰"[33]。他的父母还对外界隐瞒了一个事实：维纳在15岁之前一直不知道自己是犹太人。

离开哈佛后，维纳在剑桥随罗素学习数理逻辑，随哈代学习数论。第一次世界大战前夕，他又来到了哥廷根，随希尔伯特学习微分方程。后来，他又去了哥伦比亚大学，随约翰·杜威学习哲学。1919年，他成为麻省理工学院的教员。

在战争期间，维纳是当时新兴的通信工程和模拟计算研究领域的先锋。他参加了范内瓦·布什在MIT组织的模拟信号计算的研究项目，而且似乎这对克劳德·香农发展通信理论产生了影响。在二次世界大战期间，维纳的工作项目是研究防空火力系统。这些系统加入了比原有技术更复杂的预测部分，目标是预测飞行器为躲避导弹可能采取的路线。维纳对反馈的概念异常有兴趣，反馈就是从一个正在进行的进程中不断得到运行信息而反过来修正这个进程。

维纳并没有参加1942年5月13日在贝克曼酒店召开的第一次物理学、生物学和人类学大会，这次会议由小约西亚·梅西基金会赞助，旨在拓展各学科间的交流。麦卡洛克参加了那次会议。参加那次会议的还有人类学家格列高里·贝特森和玛格丽特·米德夫妇。维纳参加了战后的第一次梅西基金会的会议，那次会议的主题是“生物和社会科学中的反馈机制和循环因果系统”。[34]参加这次会议的还有匹茨和冯·诺依曼。这样的会议让每个人都有机会吸收他人的研究成果，并探索各自研究领域是否存在一些契合的研究目标。

1947年，维纳写了一本书，总结了这些会议上讨论的一些研究工作。他想用一个新词来描述包含了机器、生物和社会结构领域中的各种通信和反馈的研究工作。他选择了一个希腊词cybernetics，原意是“舵手”，因为舵手的本质也是利用反馈来修正航线的偏移。维纳的书出版于1948年，书名最后定为《控制论：关于在动物和机器中控制和通信的科学》（*Cybernetics: Control and Communication in the Animal and the Machine*）。

美国《时代》杂志评价道：“很少有一本书能够在很多不同的科学领域激起强烈的反响。而《控制论》就是这样的一本。”[35]今天再读起来，《控制论》是一本古怪而小巧的书，里面有很多篇幅的数学公式和令人目眩的、不太切合实际的议论。在引言里，维纳对那些启发过他的人们表示了感谢，包括麦卡洛克，“一个对研究脑皮层细胞的组织结构充满兴趣的人”；阿兰·图灵，“也许是第一个研究智能机器的逻辑可能性的人”；沃特·匹茨，“卡尔纳普在芝加哥的学生，和拉谢甫斯基及其生物物理研究团队一直保持着联系”。维纳也对计算机的先驱“哈佛的艾肯教授、高等研究院的冯·诺依曼教授、宾夕法尼亚大学负责ENIAC和EDVAC巨型机的戈登斯坦教授”表示了感谢。[36]

《控制论》的第5章是“计算机器和神经系统”。维纳对比了数字计算机的转换机制与麦卡洛克和匹茨的大脑模型：

> 众所周知，人和动物神经系统能够完成计算系统的工作。神经系统的一个显著特点是包含了适合做继电器的元素，这些元素称为

神经元或神经细胞。虽然它们在电流的刺激下会呈现复杂的特性，但是其普遍的生物特性都遵循“全或无”的原则，即要么处于休息的状态，要么当“工作”的时候经历一系列变化，而这些变化与刺激的外部环境和强度无关。[37]

两章之后，维纳写道：“认识到大脑和计算机器实现之间的共同点，可能为精神病理学甚至是精神病学的发展提供新的有效方法。”[38]然而，维纳并不是一个狂热的纯技术狂。他也很关心这个新的科学技术对人类的影响。他写了《人有人的用处：控制论与社会》（*The Human Use of Human Beings: Cybernetics and Society*），作为1948年《控制论》的补充。

控制论成为很多领域的研究焦点，这种情况一直到了1951年，维纳突然不加解释地中断了与麦卡洛克和控制论研究团队的联系。这个研究团队的人员很大程度上是因为麦卡洛克的个人魅力而聚在一起的，其中包括匹茨，他的博士论文是在维纳的指导下完成的。关于他们关系分裂有几种解释。一种观点是麦卡洛克性格较为细腻，而精神有缺陷的维纳已经无法捕捉麦卡洛克言语表达中的细微差别。有些时候，维纳分辨不出麦卡洛克是在陈述一件事实还是在说一个猜想。[39]另一种观点是维纳的妻子出于嫉妒心，为了维护丈夫的声名，谎称麦卡洛克带领的团队中有人勾引他们的女儿。[40]

作为一门统一的学科，失去了维纳和麦卡洛克联手的控制论遭受了很大的损失。在因这次决裂受到影响的人当中，匹茨可能受到的打击最大。他的精神完全垮掉了，他亲手毁掉了自己的研究成果和博士论文，开始了一段漫长的自我堕落的生活。“他不是简单地酗酒，这种人人都会的行为与他这样的高智商不匹配。他在实验室里自己合成了一种类似巴比妥类药物和鸦片的化学物质，伴着酒吞下去。”[41]匹茨于1969年死于慢性饮酒过多造成的食道静脉破裂，年仅46岁。

即便维纳和麦卡洛克之间没有分裂，也不能保证控制论会一直发展下去。在美国学术界，跨学科的概念并没有市场，专业性的研究才是取得成功的钥匙。虽然有很多复兴控制论的尝试，但是大多数只是在当今流行的词语中加入cyber前缀的文字游戏而已，就像cyborg（cybernetic organism的缩写，机械人），还有无处不在的cyberspace、cybercafe、cyberpunk和cybersex。即使是这些以cyber为前缀的词，近年来也逐渐被e开头的词取代了。

麦卡洛克和匹茨关于神经网络数学模型的论文给了冯·诺依曼很多启迪。冯·诺依曼参与了几个重大计算机项目的设计工作，包括计算机的雏形EDVAC（Electronic Discrete Variable Automatic Computer，电子

离散变量自动计算机）。在EDVAC的第一份报告（1945年6月30日）中，冯·诺依曼这样描述计算机的开关机制："每一台电子计算机都包含了起着中继作用的'元素'，这些元素有着若干个离散平衡状态，并能保持在某个状态不变。"[42]通过引用麦卡洛克和匹茨的论文，冯·诺依曼写道："值得一提的是，高级动物的神经元毫无疑问就是上述意义下的元素。"[43]

次年，冯·诺依曼开始着手研究生物和机器之间的关系。他用了一个希腊单词来表现生物体的这种特性：自动机（automaton）。[44]在一封写给维纳的信中，冯·诺依曼惊叹于他们直接研究大脑这个自然界最复杂的人工自动机是多么具有雄心壮志的一项研究工作：

> 我们的思维，我指的是你、我和匹茨的思维，目前为止都主要关注在神经系统上，更确切地说，是人类的中枢神经系统。为了搞清楚自动机及其普遍的机制，我们选择了研究大脑这个普天之下最难的问题。事实上，如果没有这些大胆而艰苦的研究工作，现阶段的研究人员，至少是我，对自动机这个课题的认识就会比现在混乱得多。这里，我也要指出，图灵在非神经领域的研究也是如此的大胆卓越。[45]

对冯·诺依曼来说，自动机就是一台有着输入、输出和中间处理过程的设备。1948年9月，他在加州理工学院的"行为的大脑机制"研讨会上做了报告。他的演讲题为"自动机的一般逻辑理论"，包含了很多关于大脑与1948年时计算机在尺寸、速度、转换机制和能源消耗上的对比。他强调了开发一种新型逻辑的必要性，并开始思考后来成为他主要课题兴趣的问题：自复制自动机。[46]

维纳对冯·诺依曼的这个想法开玩笑说："我觉得你所说的（自动机）未来能够自我复制的能力很有意思……看来有机会可以写一个新的《金西报告》[47]（Kinsey report）了。"[48]但对于冯·诺依曼来说，自复制自动机可不是一个玩笑。他一直在想是否有某种未知的规律可以阻止一台机器制造一台它的复制品。即使生物不是这样繁殖的（虽然DNA本身是自复制的），这样的问题也折射出了有趣的本体论的味道。

对自动机和图灵机的不断研究诞生了几部经典的开山之作，如通信理论的创始人香农和人工智能的先驱之一、也是Lisp语言的创造者约翰·麦卡锡合著的《自动机研究》（*Automata Studies*），由普林斯顿大学出版社在1956年出版。这本书包含了冯·诺依曼、克莱尼和人工智能先驱之一的马文·明斯基的几篇关于自动机的论文，也包含了香农和马丁·戴维斯关于图灵机的第一篇论文。

当20世纪50年代早期冷战逐渐升温的时候，维纳和冯·诺依曼发现他们各自站在了不同的政治立场上。维纳被美国在日本广岛和长崎投下的两颗原子弹的破坏力所震惊，他从此拒绝了从政府那里接受研究经费，他的文章也逐渐聚焦于现代科技的使用带来的战争和和平等社会问题。相反，冷战让冯·诺依曼产生了反共产主义的倾向，他成为核武器的坚定支持者。1955年，冯·诺依曼被诊断出骨癌，1956年开始住院治疗，次年与世长辞，终年53岁。冯·诺依曼的癌症很可能是他亲临原子弹试验场时受核辐射所致。[49]

冯·诺依曼死后留下了一系列未完成的讲义，这些讲义汇集成《计算机和人脑》（*The Computer and the Brain*）一书，于1958年出版。这本书虽然不尽人意，但它还是给了我们很多冯·诺依曼可能想表达什么内容的一些诱人的暗示。基于冯·诺依曼那些关于自动机的未完成手稿，阿瑟·W. 巴克斯编辑并完成了一本名为《自复制自动机的理论》（*Theory of Self-Reproducing Automata*）的书，于1966年出版。

在早期对自复制自动机的研究中，冯·诺依曼想象机器处在一个拥有很多备用零件的环境中，然后探讨这部机器是如何组装产生它们的复制品的。这样的自动机就是运动自动机（kinematic automata），基本上和我们平常所说的机器人是一个意思。

在与他的好友斯塔尼斯拉夫·乌拉姆（一位研究晶体增长的科学家）共同讨论后，冯·诺依曼决定先研究一个较为简单的模型——元胞自动机（cellular automata）。

元胞自动机是对于细胞结构的一个数学构造。元胞自动机可以以多种维度存在，但是实际的研究中基本上只考虑二维网格。每个网格中的细胞都会受隔壁网格中的细胞所影响，仿佛这些细胞连在了一个简单的网络中。经过持续的“移动”和“生成”，细胞根据特定规则变化不同的状态。元胞自动机的简单规则经常会产生复杂的行为。冯·诺依曼研究了拥有29个状态的元胞自动机，并且证明了这些自动机可以通过组装构成一个通用图灵机。[50]

元胞自动机在20世纪70年代突然超脱了自己的学术圈。英国数学家约翰·霍顿·康威（1937—）设计了一个他称为“生命游戏”（Game of Life）的简单元胞自动机。该自动机有一个简单规则：在一个类似方格纸的二维网格上，一个细胞要么是活的（方格被填充）要么是死的（方格未被填充）。在每一次后继的繁衍中，一个细胞根据它周围临近的8个细胞改变其自身的状态：如果一个活细胞被2个或者3个活细胞包围，它依然存活；如果被0个或1个活细胞包围，它会因孤独而死掉；被4个或4个以上活细胞包围，它也会因过度拥挤而死掉。一个被3个活细胞包围的死细胞则会因为一种“神奇”的繁衍形式而成活。

在《科学美国人》中，马丁·加德纳“数学游戏”专栏中的几个填充谜题让康威的游戏变得流行起来，[51]1974年，《时代》杂志抱怨“价值几百万美元的计算机都把时间浪费在对这种游戏持续增长的狂热中了”。[52]当然，1974年并没有个人计算机，只有大型机。现在，这个游戏大多是在个人计算机中运行了，我想《时代》杂志此时不会再说这很狂热了吧。

虽然只有简单的规则，但这台自动机却能呈现出一些非常复杂的模式，例如它可以呈现出不断繁殖后代的模式。虽然看上去不可能，但是图灵机确实可以通过这样的元胞自动机构造出来，这台自动机是图灵完备的。[53]

另一位对元胞自动机的研究感兴趣的人是德国工程师康拉德·楚泽。楚泽比图灵早两年零一天出生。当图灵在写他关于可计算数的论文时，楚泽正在他父母柏林的公寓里制造计算机。

1969年，楚泽出版了一本74页的书，名为《计算空间》（*Rechnender Raum*），是“数字物理”领域最早的一部著作。数字物理是研究如何在可计算的框架内解释宇宙运动法则的学科。

传统上，物理定律是假定为连续的。距离、速度、质量和能量的度量似乎最适合用实数表示，用微分方程运算。但是一些量子理论的观点指出，宇宙内在的自然结构可能是离散的、数字的。现实世界里自然界的连续性可能只是一个假象。“宇宙到底是数字的，还是模拟的，抑或两者皆有呢？”楚泽问道，“而提出这个问题本身是不是合理呢？”[54]为了从数字角度探索物理定律，楚泽创造了可以被元胞自动机运算的“数字粒子”的概念。《计算空间》是用数字物理描述宇宙的一个实验性的尝试，但毫无疑问，它是一个大胆的创新。

初看之下，很难把宇宙认为是一部巨型计算机。如果我们忽略相对而言非常渺小的、蜗居在宇宙中至少一个星体上的生命形式，似乎宇宙并没有涉及很多可计算的运动。那么，宇宙真的只是有一堆石头飞来飞去的时空吗？

我们从更宽阔的视角来看这个问题。现今的宇宙模型指出，宇宙开创于137亿年前的大爆炸，地球形成于45亿年前，地球上的生命始于37亿年前，最早的灵长类动物大概出现在1千万年前，而现代人类大概只能追溯到2百万年前。显然，有一些东西一直在促使着这个世界趋向于复杂。大爆炸后的最初期，宇宙是完全均匀的——一种简单的象征，然后出现较为复杂的粒子，最终原子、分子开始形成发展。这是一个由简单到复杂的过程，大概是基于相对简单的宇宙法则，这与元胞自动机有几分神似。

宇宙的计算模型一般归功于通信理论的奠基人香农和维纳对图灵机

的贡献。使用熵度量信息构建了通信学和热力学之间的桥梁，这也是过去几年一些畅销书的主题。[55]例如，麦克斯韦妖（Maxwell’s Demon），这是詹姆斯·克拉克·麦克斯韦（1831—1879）发明的一个假想精灵。用一个隔板将容器分成两格，这个精灵可以操作隔板上的一扇小门，使运动快的分子流入容器的一边，而让运动慢的分子呆在另一边，这样熵就减少了。这种想法被证明是不可行的，因为这个精灵自身也从系统中带走了部分熵。

美国物理学家约翰·阿奇博尔德·惠勒（1911—2008）把宇宙的存在形式和人类的感知联系在了一起。我们在观察的基础上问是或否的问题，并接收信息予以回答。对这个过程，用惠勒著名的三个词讲就是“万物源于比特”（it from bit）：

> “万物源于比特”象征这样一个观点：物理世界的万物从根本上，最根本上，都有非物质的来源和解释。也就是说，我们所称的现实都得于对是否问题的分析和对仪器引起的响应的记录。简而言之，所有具有物理实体的东西都源于信息论的范畴，而这就是参与的宇宙。[56]

虽然提出宇宙是由信息构建的，但是惠勒拒绝接受宇宙是任何形式的机器的概念，因为它“还得明确或隐含地假设，存在超级计算机，存在预定的计划方案，存在执行某工作的设备，存在奇迹的事件，而这就会让宇宙陷入无穷的种类和无穷的数量中”。[57]

另一个相当不同的观点来自大卫·多伊奇（1953—），他是量子计算的先驱之一。多伊奇是“多宇宙理论”最为坚定的支持人。多宇宙这个概念最早来源于美国物理学家休·埃弗莱特（1930—1982）。我们所认为的波粒二象性矛盾就是发生着不同的量子事件的多宇宙间互相干涉的结果。我们所知的这个宇宙只是多宇宙中的一个可能实例。

1997年，多伊奇出版了《宇宙的构造》。在书中，他从四个互有交织的部分来解释宇宙的本质：

〉 维也纳出生的哲学家卡尔·波普尔（1902—1994）所刻画的认识论；
〉 休·埃弗莱特在量子物理框架下的多宇宙论；
〉 英国自然学家查尔斯·达尔文（1909—1982）和生物进化学家理查德·道金斯（1941—）描述的进化论；
〉 图灵开创的计算理论。

当讨论虚拟现实生成器时，多伊奇使用了他称为“图灵原

则”（Turing principle）的概念。一开始，图灵原则似乎是关于计算机制的：“存在一种理论上的通用计算机，它可以模拟任何可能的现实物理实体的行为。”多伊奇确认这种计算机可以模拟一切物理过程。很快，多伊奇就指出了这种计算机的计算能力就等同于创造一个虚拟现实的宇宙。图灵原则逐渐演化为一个更强的版本：“创造一个虚拟现实生成器，它的所有指令包含了现实可能具有的一切环境，这是可能的。”[58]显然，这也就隐含了虚拟一个我们生活的宇宙的可行性。

MIT机械工程学教授塞思·劳埃德（1960—）更愿意将自称“量子力学”的量子物理形容为“奇异”而不是多宇宙的，但他也用计算和信息论的观点描述这个宇宙：“宇宙大爆炸也是比特大爆炸。”劳埃德拒绝宇宙可以用图灵机构建模型的观点，“宇宙本质上是量子机制的，传统的数字计算机不能模拟量子机制的系统”。[59]这也是他认为量子计算机更适合这种任务的原因之一：

> 宇宙是一个物理系统，它可以等效地由量子计算机模拟，量子计算机在尺度上和真实的宇宙一样大。因为宇宙支持量子计算，可以等效地被量子计算机模拟，所以宇宙与一部通用的量子计算机的能力并无二致……从技术角度而言，对于宇宙是否就是一部量子计算机这个问题，我们现在可以给出一个确定的答案了：是的，宇宙就是一台量子计算机。[60]

量子计算机的一个特性是传统图灵机所没有的：由量子过程产生真正的随机数的能力。

在英国物理学家、数学家、著名数学软件Mathematica的创始人斯蒂芬·沃尔夫勒姆（1959—）的作品中，元胞自动机作为一种宇宙物理定律的模型，再一次出现在公众面前，而他于2002年出版的一本极厚的、雄心勃勃而且非常畅销的书《新科学》（*A New Kind of Science*）又将此推向高潮。沃尔夫勒姆观察到元胞自动机是如何基于简单的规则而繁衍出非常复杂的模式的，受此启发，他将元胞自动机和图灵机的普遍性联系起来，以此说明它们都可以对物理过程建模。沃尔夫勒姆并没有在他的系统中引入量子机制，但是他表示他不需要引入量子机制是因为“我强烈地预感到，我讨论到的各类程序……最终都会展示出（即使不是全部，也是大部分）量子理论的主要特征”。[61]

在《新科学》中，沃尔夫勒姆在很多现实表现中找到了计算的普遍性，他定义了一种计算等价性的原则：

> 引入了一种新的自然准则，其中没有任何其他系统产生的计算

> 能比元胞自动机和图灵机所做的计算更复杂……那么，我们大脑中所进行的抽象计算又是什么呢？它们更复杂吗？答案应该是否定的，至少当我们想知道确切的结果，而非泛泛空想时是如此。如果一个计算要被显式执行，那么它一定最终会被实现为一个物理过程，因此它必然会受到其他类似物理过程所受到的限制。[62]

一旦我们相信宇宙中所有的形态都是可以计算的（不管是通过传统的数字计算机还是量子计算机），那么万事万物就要遵守这样的准则。例如，生命就是可计算宇宙的一部分，生命里最神秘的形式——人类的思维也是一样。

很多世纪以来，哲学家、生物学家、精神学家，甚至是平民百姓都一直在追寻思维的本质。我们经常认为，我们身体的大部分机能都是各种器官中一系列物理和化学过程的机械结果，我们还没有将人类思维归为此类。我们感受到的思维是如此地特别。大脑显然和思维有一定程度的关系，但是，我们同样认为大脑并不等同于思维的全部。

在西方文化里，这种思想通常称为“意识/肉体二元论”，并且通常和勒内·笛卡儿（1596—1650），尤其是他的《形而上学的沉思》（1641）联系在一起。笛卡儿相信，我们身体的大部分器官（和所有我们称之为低等动物一样）都像机器一样，但是思维不一样。

20世纪40年代，二元论遭受重大打击。对于神经学家和计算机科学家迈克尔·阿尔贝勃而言，麦卡洛克和匹茨已经在1943年的论文中解决了神经元计算的问题。大脑天然有着适合进行计算的结构，因此麦卡洛克和匹茨“展示了所有可设想的有穷计算都可以被神经网络计算出来。他们否定了二元论”。[63]

几年后，哲学家吉尔伯特·赖尔（1900—1976）在他的著作《心的概念》（*The Concept of Mind*，1949）中建立了一个很强的事实，狠狠抨击了二元论，这个事实里并没有引用麦卡洛克和匹茨的论文。今天，二元论早已黯然失色。大多数研究思维的科学家们（包括哲学家、神经学家）都默认了思维仅仅是人体物理过程，尤其是神经系统和大脑物理运作的一种表现。

在二元论渐渐被否定的同时，我们对计算和算法的认识也在不停地增长，这并不那么让人吃惊。构想中的图灵机作为人类计算机的模型，可以执行被精确定义的算法任务，所以从自动计算这一门学科诞生开始，机器和大脑之间的联系就受到关注。同样不那么让人吃惊的是最早研究人工智能概念的人中就有阿兰·图灵本人。他在其1950年最著名的论文《计算机与智能》中，发明了今天称为“图灵测试”的测验。

一旦二元论被抛弃，思维就必然被看作是大脑物理活动（协同身体

的其他部分）的一种自然表现，而不是什么超自然的东西。虽然我们在情感上有一丝排斥，但是结论是昭然的：首先，思维在能力和局限上等同于图灵机；其次，理论上完全可能制造人工的思维。

就像美国哲学家丹尼尔·丹尼特（1942—）说的："阿兰·图灵做出了基础性的开创，让我们得以将康德曾经提出的问题：怎么可能存在思维，转换成一个工程性的问题：怎么才能制造出思维。"[64]

图灵测试最让我们困扰的，同样也是大脑是计算机这种见解带来的困扰，是以第一人称一直在我们脑袋里喋喋不休的"意识"（consciousness）。意识让我们感到主观上的自主性和信仰的自由。

不过意识是难以捉摸、阴晴不定的。我们大多数人都会宣称自己在每天清醒的时候都会在心中自言自语，这让我们感到意识的存在；但当自言自语消失的时候，意识就是透明的。大多数人与人交流的时候，都假定对方有着和我们一样类似的意识，但我们并不确定，也不知道如何才能让对方意识到我们自己的意识的存在。

判断我们的大脑如何产生自我意识，是澳大利亚哲学家大卫·查默斯（1966— ）所谓的意识的"难问题"，而判断大脑如何与感知器官进行信号的输入输出，是相比而言较为简单的问题。

图灵测试（让人类测试者觉得对方像人类一样聪明）隐含着一种行为主义的观点，即不必了解个体内部是如何运作的就能将其归入或排除出"智能"的类别。我们谈论的是一种"黑盒"测试。这也是我们如何与其他人交流的方式，因为我们不能证明其他人也是有意识的。即便我们不能分辨人与机器，也非常希望能够分辨机器和我们自己。

我们所知的计算机只不过是一部遵守一套规则的机器。它们不像人类知道自己做的是什么。在这个方面，美国哲学家约翰·塞尔（1932—）做了一个著名的思想实验，也称为"汉语房间"。向一个不懂汉语的人提问，他有一本能让他给出合理答案的书，那么这个人可以通过汉语图灵测试，虽然他完全不懂问题或答案的意思。[65]

最大的问题是电脑只懂得语法，而人类还懂得语义。在塞尔看来，这说明了数字计算机（不管它会变得有多复杂）永远不能像人类一样理解它们正在做什么。

英国数学物理学家罗杰·彭罗斯（1931—）同样确信，思维不仅仅是一个计算器官的产物。在他1989年的《皇帝新脑》和1994年的《思维的影子》中，彭罗斯断言意识超出了计算的范畴。他猜测在大脑运行的是一种量子过程，这种量子过程不是算法式的，超出了图灵机的计算能力。

彭罗斯认为哥德尔的不完备性定理揭示了某种规律。我们人类能够理解哥德尔推导出的那些正确却不能证明的命题，但是任何计算都无法

证明它，因为它并不是从公理衍生出的。这不是一个新的发现，早在1958年的《哥德尔证明》（纽约大学出版社）中，欧内斯特·内格尔和詹姆士·纽曼在哥德尔的定理中找到了类似的对机器智能的反驳，同样有类似发现的还有哲学家约翰·卢卡斯（1929—）在其1961年著名的论文《心灵、机器与哥德尔》[66]中。这些论据都显示了，虽然机器能够容易地计算公理系统内的数学，但不能运行元数学，因为这需要对公理系统之外有所理解。

丹尼尔·丹尼特也许是将哲学家的深思熟虑和科学家的实证主义结合得最好的人之一，他在《意识的解释》（1991）等一些精彩的书中，对思维有着不同的描述。丹尼特吸收了可计算性的概念，并将其融合到对进化和现代神经前沿研究成果的理解当中。他眼中的大脑和思维并不是像图灵机一样的媒介：大脑是神经系统的一部分，也是身体的一部分，不能隔离地讨论大脑。大脑中有一点兴奋的想法，心跳会加快，以便让更多的氧气进入大脑。很多药物可以影响大脑。大脑从眼鼻耳等其他器官中持续接受大量的刺激，不断地通过身体与这个现实世界交流。

大脑不是一个线性处理系统，它是大规模并行分散系统，没有像“笛卡儿剧场”（丹尼特诙谐地将笛卡儿的“意识的中心”称为“笛卡儿剧场”）那样有中心作用的区域。丹尼特的大脑模型是由思维的”很多草图”构成的，包含了感官输入、视觉数据、语言等不完整的支离破碎的部分。如果大脑是一台计算机，那么这也不是一台可以由正常工程师设计出来的计算机！因为它里面一定是混乱的。

进一步讲，我们所认为的意识其实是在这种并行结构之上的一系列活动。丹尼特提出：

> 假定人类的意识：(1) 所具有的创新力是不能被硬编码在机器中的；(2) 是早期经过人类文明训练的产物；(3) 能否成功建立取决于大脑的可塑性中无数的细微设置，也就是说，意识最重要的特征对于神经解剖学来说很可能是无法剖析出来的，尽管它们起了非凡的作用。[67]

至少在某种意义上，意识会和自己“说话”，而这就需要文明的产物——语言的支持。

显然设计一台能模仿人类思维的计算机是没有意义的，这需要输入很多的数据，而且如果没有多年的训练和积累的经验，效果也不会好。不过，理论上是否可以制造一台机器可以通过无限制图灵测试呢？（丹尼特认为无限制图灵测试是很难但很公平的测试）这样的机器有意识吗？丹尼特认为这两个问题的答案都是肯定的。

不管你更倾向于大脑是以什么样的机理进行工作的，一个令人胆寒的隐含结论是，机械运作的结果决定了我们的决定，而不是其他东西。那么我们所认为的自由意识（free will）又是怎么一回事呢？

自由意识在机械化运转的宇宙中消失了，这一看法早就暗含在了决定宇宙每个粒子运动的严格的确定性法则中。皮埃尔皮西蒙·拉普拉斯（1749—1827）在他的《概率论》（*Essai Philosophique sur les Probabilites*，1814）中写道：

> 如果一个智能体理解某一时刻所有激发自然运动的力和组成宇宙的万事万物的各自状态，假如它能够在很宽广的空间分析这些数据，那么对于大到宇宙中的最大星体，小到最轻的原子，它对宇宙万事万物的运动的计算都会包含在一个公式中。对于它而言，没有什么事情是不确定的。未来就像过去一样，呈现在它眼前。[68]

这个观点通常称为拉普拉斯妖（Laplace’s Demon）。我们很难避免这一推理：在大爆炸之后，宇宙中每个原子（包括组成大脑中细胞的那些）的运动就按照一种已经确定的模式固定下来。

当然，拉普拉斯妖并不真的存在。为了跟踪宇宙中每个粒子的运动，必须用一台比宇宙自身还大的计算机存储数据。海森堡测不准原理告诉我们，基础粒子的位置和时间不能同时确定。在数学上，把研究这些原子碰撞结果的问题归类为“多体问题”（many-body problem），而即使是3体问题的计算就足够让人头疼的了。

如果宇宙确实是一台图灵机，即使我们知道当前的“完全格局”以及这个机器具有的所有格局，还是不能够预测它未来的走向，除非真正地“跑”一遍“程序”。

不确定性是自由意识的基础。塞思·劳埃德指出，

> 停机问题不仅适用于传统的数字计算机，也适合于能进行数字逻辑运算的系统。因为粒子碰撞本质上进行的是数字逻辑的计算，所以它们的未来是不可计算的……我们面临抉择的时候所感知的主观随意性就类似停机问题：一旦我们脑中有一些想法，我们并不知道它会引领我们走向何方。即使它确实引领我们去了某个地方，在到达之前，我们也不知道是在哪里。[69]

大卫·多伊奇仔细考虑了大脑是“经典”的非量子计算机，而不是量子计算机的可能性：

> 都说大脑可能是一台量子计算机，而且直觉、意识和我们解决

问题的能力都基于量子计算。这可能是正确的，但是我没有看到任何证据或任何让人信服的论据，证明这是正确的。我的看法是，大脑如果被认为是计算机，那么它是一台“经典”的计算机。[70]

然后他承认，“图灵对于计算的解释，即使是从原理上，也似乎仅仅给从物理角度进一步探索诸如意识和自由意识等精神属性留下了很小的空间”。记住，在量子物理的多宇宙理论里，世界是不断分裂的，在一个世界里你可以选择做这件事，而在另一个世界里你也可以选择做另一件事。如果这都不是自由意识，那么什么才是呢？多伊奇总结道：”图灵对于计算的概念似乎与人类的价值观不太相关，在多宇宙的框架下理解这些，对于我们认识人类主观意识等精神属性并没有阻碍。”[71]

斯蒂芬·沃尔夫勒姆在研究元胞自动机表现出的复杂结构时，他试图寻找预测结果的方法，或者至少能够找到可以减小繁衍代数而保持结果不变的捷径。但是他不能，”完全无法预测系统将会如何表现，除非像系统自身进化的过程那样一步一步地计算……对于很多系统，根本无法进行系统性的预测，也没有普遍意义上的进化捷径……”不可能进行有效预测这一事实给了系统以行使自由意识的自由，沃尔夫勒姆甚至还给出了一个图表，展示了一个”行为表现出类似自由意识的元胞自动机”。[72]

这也算是一种慰藉了。即使宇宙和大脑像元胞自动机和图灵机一样，都是以一套简单的规则为基础，并繁衍出复杂的结构，我们依然无法基于这些规则预测未来。在我们运行到属于未来的那一行代码前，它并不存在。

就像在《回到未来》三部曲[73]的结尾中，布朗博士对麦克弗莱和帕克说的那样：“这说明你们的未来还未书写，每个人的未来都还未书写。你们的未来取决于你们如何打造它。所以你们二人要好好把握。”

---

[1] 这些层次可参见本书作者所著的*Code: The Hidden Language of Computer Hardware and Software*（Microsoft Press，1999）。

[2] 斯蒂芬·科尔·克莱尼，“Origins of Recursive Function Theory”，*Annals of the History of Computing*， Vol. 3, NO.1（Jan.1981），61。

[3] 斯蒂芬·科尔·克莱尼，*Introduction to Metamathematics*（D. Van Nostrand，1952），358-360。

[4] 克莱尼，*Introduction to Metamathematics*，358。

[5] 克莱尼，*Introduction to Metamathematics*，300，376。

[6] 克莱尼，*Mathematical Logic*（John Wiley & Sons，1967；Dover，

2002），232。

[7] 参见B. Jack Copeland，*The Essential Turing*（Oxford University Press，2004），40，脚注61。

[8] 马丁·戴维斯，*Computability and Unsolvability*（McGraw-Hill，1958，Dover，1982），vii-viii。

[9] 马丁·戴维斯，*Computability and Unsolvability*，vii。

[10] 马丁·戴维斯，*Computability and Unsolvability*，70。

[11] Raúl Rojas，“How to Make Zuse’s Z3 a Universal Computer”，*IEEE Annals of the History of Computing*，Vol. 20，No. 3（1998），51-54。

[12] 阿兰·图灵，“Systems of Logic Based on Ordinals”，*Proceedings of the London Mathematical Society*，*Series* 2，Vol. 45（1939），172-173。

[13] 马丁·戴维斯，“The Myth of Hypercomputation”，Christof Teuscher，ed., *Alan Turing*：*Life and Legacy of a Great Thinker*（Springer, 2004），195–211。

[14] 参见C. Jack Copeland，“The Church- Tuning Thesis”里面有准确的评论。Stanford Encyclopedia of Philosophy，http: //plato.stanford.edu/entries/church-turing。

[15] 这个故事来自Pamela McCorduc所著的*Machines Who Think A Personal Inquiry into the History and Prospects of Artificial Intelligence* [25th anniversersary edition（A. K. Peters，2004）89]一书中，由麦卡洛克的前学生Manual Blum所述。它也出现在由麦卡洛克的夫人Rook McCulloch编辑的*Collected Works of Warren S. McCulloch* [Intersystems Publications，1989， Vol. I， 31]一书Manual Blum的文章“Notes on McCorduc-Pitts’ *A Logical Calculus of the Ideas Immanent in Nervous Activity*”中 。故事的另一版本还详述于James A. Anderson和Edward Rosenfeld所编*Talking Nets: An Oral History of Neural Networks* [MIT Press，1998，218]中对Michael A. Arbib的访谈。

[16] 采访自Jack D. Cowan *Talking Nets*，104。

[17] 采访自Jerome Y. Lettvin，*Talking Nets*，2。参见Jerome Y. Lettvin，“Warren and Walter”，*Collected Works of Warren S. McCulloch*，Vol. II，514-529，匹茨在图书馆开放时间用了一星期阅读《数学原理》。

[18] Neil R. Smalheiser “Walter Pitts”，*Perspectives in Biology and Medicine*，Vol.43，No. 2 （Winter 2000），218。

[19] Smalheiser，“Walter Pitts”， 22。

[20] 4F级别指“精神或身体上不适合服兵役”的一类人群。——译者注

[21] 同⑤。

[22] Tara H. Abraham，“Nicholas Rashevsky’s Mathematical Biophysics”，*Journal of the History of Biology*，Vol.37，No.2（Summer 2004），333–385。
[23] 沃伦·麦卡洛克，“What is a Number，That a Man May Know it, and a Man, That he May Know a Number ?”，*Collected Words of Warren S. McCulloch*，Vol. IV，1226。
[24] 麦卡洛克的很多传记信息来自Michael A. Arbib，“Warren McCulloch’s Search for the Logic of the Nervous System”，*Perspectives in Biology and Medicine*，Vol.43，No.2（Winter 2000），193–216。
[25] 采访自Jack D. Cowan，*Talking Nets*，102。
[26] Arbib，“Warren McCulloch’ s Search for the Logic of the Nervous System”，202。
[27] Arbib，“Warren McCulloch’s Search for the Logic of the Nervous System，”199。
[28] Tara H. Abraham，“（Physio）logical Circuits：The Intellectual Origins of the McCulloch-Pitts Neural Networks，” *Journal of the History of the Behavioral Sciences*，Vol.38，No.1（Winter 2002），19。
[29] 麦卡洛克和匹茨，“A Logical Calculus in the Ideas Immanent in Nervous Activity”，*Bulletin of Mathematical Biophysics*，Vol. 5（1943），129。也可参见 *Collected Works of Warren S. McCulloch*，Vol. I，357。
[30] 麦卡洛克Lloyd A. Jeffress，ed.，*Cerebral Mechanisms in Behavior: The Hixon Symposium*（John Wiley & Sons, 1951），32–33。
[31] 麦卡洛克，“Mysterium Iniquitatis of Sinful Man Aspiring into the Place of God”，*The Scientific Monthly*，Vol. 80，No.1（Jan.1955），36。也见*Collected Works of Warren S. McCulloch*，Vol. III，985。
[32] 同上，38。
[33] Flo Conway and Jim Siegelman，*Dark Hero of the Information Age: In Search of Norbert Wiener, the Father of Cybernetics*（Basic Books，2005），21。
[34] Conway and Siegelman，*Dark Hero*，155。
[35] December 27, 1948 issue, 引自Conway and Siegelman，*Dark Hero*，182。
[36] 维纳，*Cybernetics: or Control and Communication in the Animal and the Machine* （John Wiley & Sons，1948；second edition，MIT Press，1961），12，13，14，15，页码出自第2版。
[37] 维纳，*Cybernetics*，120。

[38] 维纳，*Cybernetics*，144。
[39] Arbib，“Warren McCulloch’s Search for the Logic of the Nervous System，” 201-202。
[40] Conway and Siegelman，*Dark Hero*，222–229。
[41] Smalheiser，“Walter Pitts，”223。
[42] 冯·诺依曼，*First Draft of a Report on the EDVAC*（Moore School of Electrical Engineering，1945），§4.1。
[43] 同上，§4.2。
[44] William Aspray，“The Scientific Conceptualization of Information: A Survey”，*Annals of the History of Computing*，Vol. 7，No. 2（April, 1985），133。
[45] Letter of November 29，1946，from Miklós Rédei，ed.，*John von Neumann: Selected Letters* （American Mathematical Society，2005），278。
[46] 冯·诺依曼，“The General and Logical Theory of Automata”in Lloyd A. Jeffress，*Cerebral Mechanisms in Behavior: The Hixon Symposium*（John Wiley & Sons，1951），1-41。
[47]《金西报告》原名《人类男性的性行为》，是世界性学史上的一部里程碑式的著作。在其广为流传后，人们更喜欢用作者的名字来称呼它。金西指作者阿尔弗雷德·金西，报告指它是一部性社会学意义上的调查报告和资料汇编。——译者注
[48] Letter of August 10，1949，quoted in Steve J. Heims，*John von Neumann and Norbert Wiener: From Mathematics to the Technologies of Life and Death*（MIT Press，1980），212。
[49] Heims，*John von Neumann and Norbert Wiener*，369-371。
[50] William Aspray，*John von Neumann and the Origins of Modem Computing*（MIT Press，1990），203-204。
[51] 这种游戏后来被马丁·加德纳收录，*Wheels, Life, and Other Mathematical Amusements*（W. H. Freeman，1983）。
[52] 1974年1月21日发行，引自William Poundstone，*The Recursive Universe*：*Cosmic Complexity and the Limits of Scientific Knowledge*（William Morrow，1985），24。
[53] Paul Rendell，“A Turing Machine in Conway’s Game of Life”，March 8，2001。http://www.cs.ualberta.ca/~bulitko/F02/papers/tm_words.pdf。也可见http://rendell-attic.org /gol/tm.htm。
[54] 康拉德·楚泽，*Calculating Space*，译自*Rechnender Raum*（MIT Technical Translation，1970），22，（德文本p16）。

[55] Tom Siegned，*The Bit and the Pendulum: From Quantum Computing to M Theory — The New Physics of Information*（John Wiley & Sons，2000）。Hans Christian von Baeyer，*Information: The New Language of Science*（Harvard University Press，2003）。Charles Seife，*Decoding the Universe: How the New Science of Information is Explaining Everything in the Cosmos from Our Brains to Black Holes*（Viking，2006）。
[56] 约翰·阿奇博尔德·惠勒，“Information, Physics, Quantum: The Search for Links”（1989） in Anthony J. G. Hey，ed.，*Feynman and Computation: Exploring the Limits of Computers*（Perseus Books，1999），311。
[57] 同上，314。但是惠勒引用了他写自1988年的另一篇论文。
[58] 大卫·多伊奇，The Fabric of Reality（Penguin Books，1997），132-135
[59] 塞思·劳埃德，*Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos*（Alfred A. Knopf，2006），46，53。
[60] 同上，54-55。
[61] 斯蒂芬·沃尔夫勒姆，*A New Kind of Science*（Wolfram Media，2002），538。
[62] 同上，720，721。
[63] Arbib，“Warren McCulloch’s Search for the Logic of the Nervous System”，213。
[64] 丹尼尔·丹尼特，interview in Susan Blackmore，*Conversations on Consciousness: What the Best Minds Think About the Brain, Free Will, and What it Means to be Human*（Oxford University Press，2006），81。
[65] 约翰·塞尔，“Minds, Brains, and Programs，”from *The Behavioral and Brain Sciences*，Vol. 3（Cambridge University Press，1980。Republished in Douglas R. Hofstandter and Daniel Dennett，eds.，*The Mind’s I: Fantasies and Reflections on Self and Soul*（Basic，Books，1981），353-373。
[66] 约翰·卢卡斯，“Minds, Machines and Gödel”，*Philosophy*，Vol. 36，No.137（Apr.-Jul. 1961），112-127。
[67] 丹尼尔·丹尼特，*Consciousness Explained*（Back Bay Books，1991），219。
[68] 皮埃尔-西蒙·拉普拉斯，*A Philosophical Essay on Probabilities*，译者Frederick Wilson Truscott 与 Frederick Lincoln Emory（John Wiley & Sons，1902，Dover，1995）。
[69] 劳埃德，*Programming the Universe*，98，36。

[70] 丹尼特，*The Fabric of Reality*，238。
[71] 同上，336，339。
[72] 沃尔夫勒姆，*A New Kind of Science*，739，741，750。
[73] 编剧鲍勃·盖尔，根据其与罗伯特·泽米吉斯的故事和人物改编。

第18章 长眠的丢番图

图灵和邱奇证明了，不存在通用的过程来判定任一命题在一阶谓词逻辑系统是否可证。然而，这之后很久，最初的判定性问题依然没有解决。这就是著名的希尔伯特第10问题，它是大卫·希尔伯特于1900年在巴黎国际数学大会的演讲中提出的20世纪数学家面临的最重要数学问题中的一个：

> 10．丢番图方程可解性的判定。
>
> 给定一个包含任意个未知数的有理整系数不定方程，试推导一个过程，通过有限步运算判定该方程是否存在有理整数解。[1]

公元3世纪，亚历山大数学家丢番图在他的《算术》中所提到的代数问题，都可以用一些整系数多变量多项式来描述。希尔伯特的第10问题就是要求一个通用过程来判定某个特定的丢番图方程是否有整数解。

诚然，在哥德尔的不完备性定理以及邱奇和图灵的不可判定结果诞生以后，很少有数学家还期望有人能达成希尔伯特的愿望，“设计一个过程”来判定丢番图方程的可解性。很多人其实期望的是一种相反的结果：不可能有这种通用过程的证明。

很多数学家深深着迷于希尔伯特的第10问题，其中有一个人，她将自己整个职业生涯都投入到了追寻第10问题不可解的证明中。这个人就是朱莉亚·罗宾逊。

朱莉亚·罗宾逊最早的记忆，是她在亚利桑那州家附近的巨型仙人掌边摆弄鹅卵石。

> 我想我天生就喜欢研究自然数。对我来说，它们就是实实在在的事物。我们可以想象与我们现在迥然不同的化学或生物学科，但是无法想象另一个跟现在不同的数字数学。在任一宇宙中，数字的定理都是通用的。[2]

朱莉亚·罗宾逊于1919年出生在圣路易斯州，原名朱莉亚·鲍曼。她

有一个大她2岁的姐姐康斯坦斯。当朱莉亚·罗宾逊2岁的时候，她的妈妈去世了，她们搬到了凤凰城附近的祖母家里，后来又搬到了圣迭戈湾附近的洛马岬，与父亲及继母一起住。

朱莉亚9岁的时候，得了猩红热，然后又是风湿热，因此休学了2年。为了帮她赶上学业进度，父母聘请了家教，让朱莉亚在一年内补习了从5年级到8年级的功课。

> 有一天她告诉我，无法将2的平方根计算到小数点后的某一位起开始循环。她知道，这已是证明过的，虽然她不知道是如何证明的。我不明白这是怎么证明的，所以回到家，利用新学到的一些开平方的技巧试图自己证明它，但是到那天下午最终放弃了。[3]

1933年，朱莉亚进入了圣迭戈高中。这一年，大批的数学家开始陆续逃离哥廷根和其他德国高等学府。朱莉亚童年时的疾病让她变得害羞、安静、自卑，但也造就了她独自工作时的耐心和坚毅。

上到高年级的时候，她成了班上唯一既选修物理又选修数学的女生，而且这些科目她都能获得好成绩。她收到的高中毕业礼物是一把计算尺。1936年秋，未满17岁的朱莉亚去了圣迭戈大学（现更名为圣迭戈州立大学），主修数学。那时，她的学费是每学期12美元。朱莉亚希望自己将来成为一名教师，“那时我没想过要当数学家（数学家毕竟和数学教师完全不同）”。[4]

1937年，Simon & Schuster公司出版了一本非常有名的书《数学精英》（*Men of Mathematics*），作者是数学家E. T. 贝尔（1883—1960）。读了这本书，朱莉亚才知道数学家是什么，他们真正做的是什么。虽然这本书在历史和人物个性方面有虚构的成分，但是给了朱莉亚从未有过的鼓舞，就像此后它给其他很多同样处于豆蔻年华的数学家带来的一样。

1939年，朱莉亚离开了圣迭戈州立大学，去了加州大学伯克利分校。当时伯克利正在筹建一个强大的数学系。在第一学年，她的数论老师是拉斐尔·罗宾逊，仅大她8岁。“第二学期，我们仅有4名学生，同样，我是唯一的女生。拉斐尔让我陪他散步……在一次清晨散步的路上，他向我介绍了哥德尔的研究成果。”[5]

也许他们还谈论了很多其他理论。1941年，拉斐尔·罗宾逊和朱莉亚·鲍曼结婚了。由于有避免裙带关系的规定，因而朱莉亚不能在伯克利的数学系任教，虽然她已经在统计系得到了助教一职。（当她申请这份工作时，人事部门问她每天都做什么。她写道：“星期一——证明命题，星期二——证明命题，星期三——证明命题，星期四——证明命

题，星期五——命题是错的。”[6]

虽然拉斐尔和朱莉亚都想要个孩子，但是朱莉亚的童年疾病让她的内心变得十分脆弱。她有过一次流产，医生强烈建议她打消要孩子的念头。[7]

在1946～1947这一学年，拉斐尔以访问学者身份去了普林斯顿大学。他和朱莉亚上了由邱奇主讲的课程，还在纪念普林斯顿大学二百周年期间听了哥德尔关于数学基础的课。

回到伯克利后，朱莉亚在著名的波兰裔犹太逻辑学家阿尔弗雷德·塔斯基（1902—1983）指导下攻读博士，并在1948年拿到了博士学位。就是在那时，她的博士论文揭示了“她的主要学术兴趣在逻辑学和数论交织的领域”。[8]

1948年，塔斯基让她研究一个问题，这个问题牵涉到另一个问题，而这就是决定了她作为数学家的职业生涯的希尔伯特第10问题。

就像大多数数学家一样，朱莉亚并没指望第10问题存在令希尔伯特满意的解。在第一篇关于丢番图方程的论文[9]中，她写道：

> 鉴于针对很多经典的带一个未知数的丢番图方程，还没有有效的方法判定其对于任意参数值的可解性，因而很可能不存在一个判定过程。例如，还没有方法判定以下丢番图方程是否可解：
>
> $$x^2 + ay^2 = s^2, x^2 - ay^2 = t^2,$$
>
> （早在中世纪，阿拉伯人就研究过这个丢番图方程。）

一些数学家倾向于迂回解决丢番图方程。他们定义了丢番图集合，集合包含了某一特定丢番图方程的所有解。例如，在下面的丢番图方程中，所有偶数的集合就是该方程所有整数解x的集合：

$$x - 2y = 0$$

这个方程有2个变量，但是丢番图集合只由x的取值构成。如果x和y都是整数，那么x一定是偶数。

对于含有多变量的丢番图方程，可以定义一个丢番图关系（Diophantine relation）。例如，假设你想表达x比y小这一关系，满足这个关系的x和y的取值就是下面丢番图方程的解：

$$x - y + z + 1 = 0$$

朱莉亚的论文并没有证明这些集合与关系就是所谓的丢番图集合与丢番图关系，但她指出这些集合和关系可以用指数来定义，也就是x^y，

其中x和y都是变量。

将指数引入丢番图方程的讨论中一开始看上去并不相关。丢番图方程中不允许指数出现。丢番图方程可以包含变量的整数次幂，但是不能包含变量的变量次幂。

不过，这篇论文指出指数是一个很重要的关系，因为二项式系数、阶乘函数和素数集合都可以用指数来定义。指数关系有没有可能就是丢番图关系，因为它可以定义为丢番图关系？答案并不肯定，但是朱莉亚的论文还证明了，指数可以用任何一个有着近似指数级增长的丢番图关系来定义。虽然目前还未发现这样的丢番图关系，但她说“非常有可能”[10]存在这样的丢番图关系。

严格来说，费马的最后定理（也称费马大定理）并不是一个丢番图方程：

$$x^n + y^n = z^n$$

费马定理声称这个方程在n大于2时没有整数解，也就是说，n在这里被视为变量。将n替换成任意一个大于2的整数，这个方程就变成了一个丢番图方程。如果指数关系确实是一种丢番图关系，那么费马的这个方程就成为一个标准的丢番图方程，虽然它比一般形式更为复杂。

朱莉亚的论文发表在1950年的国际数学家大会上。那届大会在哈佛召开，也是在那里，朱莉亚第一次遇见了马丁·戴维斯。

马丁·戴维斯早在纽约城市学院读本科的时候，就为希尔伯特的第10问题所着迷。纽约城市学院数学系的教授埃米尔·波斯特写过一句话，大意是第10问题“正乞求人们证明它是不可解的”。[11]

戴维斯在普林斯顿读研究生的时候，发现自己“完全没办法不想希尔伯特的第10问题，我认为，对于这个如此困难的问题我几乎不可能看到出路，我试着将自己的注意力从它身上移开，却发现做不到”，虽然戴维斯的博士论文确实涉及了这个主题。

马丁·戴维斯参加国际数学大会时刚刚拿到博士学位。朱莉亚还记得马丁·戴维斯当时对她论文的反应有点让人费解：“我记得他说过，他并不觉得我的论文对解决希尔伯特问题有什么帮助，因为论文里只是一系列的例子。我说，好吧，我已经尽力了。”[12]戴维斯后来承认：“我确实跟她说过我怀疑她的做法偏离了目标，这是我一生中说过的最愚蠢的话之一。”[13]

朱莉亚和她的姐姐康斯坦斯自从长大后就很少在一起，但在1950年，康斯坦斯嫁给了内尔·雷德，旧金山大学的法律系学生。从此她们住得很近并经常来往，关系变得非常密切。在罗宾逊夫妇的鼓励下，康

斯坦斯写了她的第一本书《从0到无穷》（*From Zero to Infinity*，1955）。在她们姐妹去了哥廷根“朝圣”后，康斯坦斯又写了一本关于希尔伯特的传记（1970）。我在本书的第3章中大量参考了这本传记。康斯坦斯随后又写了几本传记，她笔下的人物有理查德·柯朗（1976）、奈曼（1982）、 E. T. 贝尔（1993）。后来，她专门写了一本书献给妹妹：《朱莉亚：数学人生》（*Julia: A Life in Mathematics*，1996）。

在1958年、1959年和1960年的夏天，马丁·戴维斯与希拉里·普特南（1926—）一起共事。普特南更为人们熟知的身份是哲学家，但是他在数学和哲学方面的造诣都很深。1959年夏，他们开始将朱莉亚的方法引入到工作中。最后，他们寄给朱莉亚一份其论文的草稿。朱莉亚改进了其中的一部分，三个人联名完成了这份新的论文“指数丢番图方程的判定问题”（The Decision Problem for Exponential Diophantine Equations），并于 1961年发表。[14]

就如论文标题，这篇论文是关于指数丢番图方程的，即丢番图方程的一种变形，它允许指数以几种形式出现：变量的变量次幂，常数的变量次幂，变量的常数次幂（也就是一般的丢番图方程）。戴维斯戴普特南普朱莉亚合作的这篇论文，在第二句话提到：证明的结果是“并没有一般的算法来判定指数丢番图方程是否有正整数解”。

现在，离否定希尔伯特第10问题仅一步之遥，这关键的一步就是戴维斯称之的“朱莉亚·罗宾逊假定”。[15]这个假定是说存在近似指数级增长的丢番图关系，这也蕴涵了指数关系本身就是丢番图关系，也就是说，指数丢番图方程可以表示为一般的丢番图方程。

20世纪60年代，朱莉亚一直是伯克利的讲师，讲授数学（其中一个学期讲授的是哲学），偶尔从事和发表一些希尔伯特第10问题的研究。在1969年的《数论的研究现状》（*Studies in Number Theory*）中，她写了40页篇幅的一章来总结当时数论研究的进展，这章提出了一个最为重要的难题：

> 关系 $r = S^t$ 是丢番图关系吗？如果是，所有的指数丢番图方程都可以替换成等价的有更多变量的丢番图方程。另外，所有的递归可枚举关系也都是丢番图关系，因此希尔伯特的第10问题不可判定。目前为止，我们依然不知道是不是这样。[16]

20世纪60年代，戴维斯在雷舍利尔理工学院和纽约大学授课，经常有机会讲授希尔伯特的第10问题。如果有人请他预测这个问题的可解性或不可解性，他就会像希伯来圣经里的先知那样，说出早已准备好的回答：“我认为朱莉亚·罗宾逊假定是成立的，而且它会被一个聪明的年轻

俄国人证明。”[17]

尤里·马蒂亚塞维奇于1947年出生在俄罗斯的圣彼得堡（即前苏联的列宁格勒），他高中时候的学校很注意学生数学和科学素质的培养。马蒂亚塞维奇17岁的时候进入国立列宁格勒大学学习，1966年就在莫斯科进行的国际数学家大会上发表过演讲。1970年，他从斯捷克洛夫数学研究所列宁格勒分部（也就是著名的LOMI）取到了博士学位。

马蒂亚塞维奇第一次听说希尔伯特第10问题是在1965年，当时他在国立列宁格勒大学读本科二年级。他的导师马斯洛夫（1939—1981）轻描淡写地告诉他：“试着去证明丢番图方程的不可解性。这个问题也叫做希尔伯特第10问题，但你不必理会这些。”他还向马蒂亚塞维奇推荐了一种研究方法：“现在，不可解证明的通常途径是，将需要证明的问题规约到已知证明的其他不可解的问题上。”马斯洛夫不推荐马蒂亚塞维奇看关于希尔伯特第10问题的相关研究文献，他说：“关于这个问题，现在只有几个美国数学家发表了一些研究成果，你不需要看……美国人至今还没证明这个问题，所以他们的方法很可能不恰当。”[18]

几年里，马蒂亚塞维奇尝试了一些不同的方法解决希尔伯特第10问题，但都陷入了死胡同。他对这个世界难题的痴迷逐渐传遍了国立列宁格勒大学的校园。一位教授嘲笑他：“你证明了希尔伯特第10问题不可解了吗？还没有？那么你将无法从这里毕业！”[19]马蒂亚塞维奇最后决定读一读美国人的论文，这其中包括了最重要的1961年戴维斯年普特南普朱莉亚合著的论文。

1970年新年刚过，马蒂亚塞维奇找到了一个与斐波那契数有关的满足朱莉亚·罗宾逊假设的丢番图关系。他当年只有22岁。他在1月底做了关于希尔伯特第10问题不可判定的第一次公开报告，很快传遍了世界。朱莉亚·罗宾逊写信给他：“让我特别高兴的是，如果你真的才22岁，那当我最初提出那个猜想的时侯，你还是个孩子，而我不得不等你长大!“[20]

朱莉亚和拉斐尔·罗宾逊于1971年前往列宁格勒，拜访了马蒂亚塞维奇夫妇。在接下来的几年间，他们通过邮件在丢番图方程的问题上又合作出了几篇文章。

马丁·戴维斯在《科学美国》杂志[21]上就希尔伯特第10问题写了一篇脍炙人口的文章。1982年，多佛出版社再版经典著作《可计算性和不可判定性》时，他又写了一篇更有技术性的文章作为这本书的附录二。1974年5月，美国数学学会在北伊利诺伊斯大学举行了一场纯数学的座谈会，这次会议专注于希尔伯特问题。戴维斯、马蒂亚塞维奇和朱莉亚共同呈现了一篇文章：”丢番图方程：负面解的积极因素“（Diophantine Equations: Positive Aspects of a Negative Solution）[22]，在其中探讨了

从丢番图方程不可解的证明中推导出来的几个有用结论。

由于在解决希尔伯特第10问题中起到的杰出作用，朱莉亚·罗宾逊广为人知。1976年，她终于在伯克利晋升为正教授，并成为美国科学院第一位女性数学家。1982年，她成为美国数学学会第一位女主席。她还被《家庭妇女期刊》（*Ladies Home Journal*）列入百位美国最杰出女性的名单。[23]

1983年，朱莉亚·罗宾逊被授予麦克阿瑟奖（也称为“天才奖”）。她匿名捐赠了部分奖金，用来支持牛津出版社出版《哥德尔文集》。这本书现在是每一个研究数学逻辑和可计算性的学者必参考的一本图书。

1984年，朱莉亚·罗宾逊被诊断出患有白血病。她于次年逝世，享年65岁。她的丈夫拉斐尔·罗宾逊在1995年去世，享年83岁。

马蒂亚塞维奇现在已是花甲之年。本书出版时，马丁·戴维斯也有80岁了。他们两个人现在仍然活跃在数学界。

1993年，马蒂亚塞维奇写了《希尔伯特的第10问题》（*Hilbert's Tenth Problem*），很快被麻省理工出版社译为英文出版。虽然希尔伯特第10问题不可解的证明占据了这本书的前一百页，但马蒂亚塞维奇重写了证明，而这个重写后的证明自成体系，几乎无需引用任何之前的研究。

《希尔伯特的第10问题》的第5章提到了图灵机。作为计算机时代的产物，马蒂亚塞维奇用现代编程语言的专业术语描述他的机器的工作原理，并阐述了它们和程序设计语句的关系。他证明了“图灵机不能判定某一类丢番图方程是否有解，更不要说任意丢番图方程了”。[24]

那个古老的谜语告诉我们：丢番图的童年占一生的六分之一；过了十二分之一，两颊长须；又过了七分之一，找到了终生伴侣；五年之后，婚姻之神赐给他一个儿子，可儿子享年仅是其父的一半就进入了冰冷的坟墓；悲伤只有靠研究数学问题去消解，又过四年，他也走完了人生的旅途。

17个世纪后，阿兰·图灵在与丢番图儿子相似的年纪，溘然长逝。后人借助他用无与伦比的想象力创造出来的翅膀才得以继续探索人类智慧的潜力和局限，并追求人类智慧在逻辑和数学上的意义。

丢番图和图灵死后都留下了谜题。就像人类的动机一样，一些丢番图方程有解，一些没有，而许多其他事情，我们永远都不知道。

---

[1] Ben H. Yandell，*The Honors Class: Hilbert's Problems and Their Solvers*（A. K. Peters，2002），406。

[2] 康坦坦斯· 雷德，*Julia*：*A Life in Mathematics*（Mathematical Association of America，1996）， 3。这段引自题为“The Autobiography

of Julia Robinson，”的书，这是由康斯坦斯撰写的，基于她对自己的妹妹朱莉亚的采访，最初发表在康斯坦斯，*More Mathematical People*（Academic Press，1990），262–280一书中。

[3] 康斯坦斯，*Julia: A Life in Mathematics*，9。

[4] 同上，21。

[5] 康斯坦斯，*Julia: A Life in Mathematics*，31，35。

[6] 同上，33。

[7] 同上，43。

[8] 康斯坦斯· 雷德和拉斐尔· 罗宾逊，“Julia Bowman Robinson（1919—1985）”，in *A Century of Mathematics in America*，Part III（Amencan Mathematical Society，1989），410。

[9] 朱莉亚·罗宾逊，“Existential Definability in Arithmetic”，*Transactions of the American Mathematical Society*，Vol. 72（1952），437-449。Also published in Julia Robinson，*The Collected Works of Julia Robinson*，ed Solomon Feferman（American Mathematical Society，1996），47-59。

[10] 罗宾逊，“Existential Definability in Arithmetic”，438。

[11] 马丁·戴维斯，Foreword to Yuri V. Matiyasevich，*Hilbert’s Tenth Problem*（MIT Press，1993），xiii。

[12] 康斯坦斯·雷德，*Julia: A Life in Mathematics*，61。

[13] 马丁·戴维斯，*Hilbert’ s Tenth Problem* 序言，xiv。

[14] 戴维斯—普特南—朱莉亚，“The Decision Problem for Exponential Diophantine Equations”，*Annals of Mathematics*，Vol. 74，No.3（Nov. 1961），425-436。也包含在 *The Collected Works of Julia Robinson*，77-88。

[15] 戴维斯，*Hilbert’s Tenth Problem* 序言，xiii。

[16] 朱莉亚·罗宾逊，“Diophantine Decision Problems”，W. J. LeVeque，ed.，*Studies in Number Theory*（Mathematical Association of America，1969），107。也参见 *Collected Works of Julia Robinson*，176。

[17] 戴维斯， *Hilbert’s Tenth Problem* 序言，xiii。

[18] 尤里·马蒂亚塞维奇，“My Collaboration with Julia Robinson，” *The Mathematical Intelligencer*，Vol. 14 ，No. 4（1992），38-45。转载于康斯坦斯·雷德，*Julia: A Life in Mathematics*，99-116。

[19] 尤里·马蒂亚塞维奇，“My Collaboration with Julia Robinson，” *The Mathematical Intelligencer*，Vol. 14 ，No. 4（1992），38-45。转载于康斯坦斯·雷德，*Julia: A Life in Mathematics*，99-116。

[20] 康斯坦斯·雷德，*Julia: A Life in Mathematics*，73。

[21] Martin Davis and Reuben Hersh，“Hilbert’ s 10th Problem”，*Scientific*

American，Vol. 229，No.5（Nov. 1973），84-91。
[22]戴维斯、马蒂亚塞维奇和朱莉亚，“Hilbert’s Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution”，in Felix E. Browder，ed.，*Mathematical Developments Arising from Hilbert Problems*（American Mathematical Society，1976），Vol. 2，323-378。也可参见 *Collected Works of Julia Robinson*，269-324。
[23]康斯坦斯，*Julia: A Life in Mathematics*，81。
[24]马蒂亚塞维奇，*Hilbert’s Tenth Problem*，93。

参考文献

网络资源

阿兰·图灵网站由安德鲁·霍奇斯维护，网址是www.turing.org.uk。
计算历史的图灵档案，网址是www.AlanTuring.net。
图灵数字档案，网址是www.turingarchive.org。

书籍

Ackermann, Wilhelm. *Solvable Cases of the Decision Problem.* North-Holland Publishing Company, 1962.

Agar, John. *Turing and the Universal Machine: The Making of the Modern Computer.* Icon Books, 2001.

Arbib, Michael A. *Brains, Machines, and Mathematics.* McGraw-Hill, 1964.

Aspray, William. *John von Neumann and the Origins of Modern Computing.* MIT Press, 1990.

Beth, Evert W. *The Foundations of Mathematics: A Study in the Philosophy of Science,* second edition. North-Holland, 1964; Harper & Row, 1966.

Boolos, George S., John P. Burgess, and Richard C. Jeffrey. *Computability and Logic,* fourth edition. Cambridge University Press, 2002.

Börger, Egon, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem.* Springer, 1997, 2001.

Boyer, Carl B. *A History of Mathematics,* second edition revised by Uta C. Merzbach. John Wiley & Sons, 1989.

Cantor, Georg. *Contributions to the Founding of the Theory of Transfmite Numbers,* translated and with an introduction by Philip E. B. Jourdain. Open Court, 1915; Dover, 1955.

Carpenter, B. E. and R. W. Doran. *A. M. Turing's ACE Report of 1946 and Other Papers.* MIT Press, 1986.

Ceruzzi, Paul E. *A History of Modern Computing*, second edition. MIT Press, 2003.

Ceruzzi, Paul E. *Reckoners: The Prehistory of the Digital Computer, from Relays to the Stored Program Concept, 1935-1945.* Greenwood Press, 1983.

Chaitin, Gregory. *Meta Math! The Quest for Omega.* Vintage Books, 2005.

Chaitin, Gregory. *Thinking about Gödel and Turing: Essays on Complexity, 1970-2007.* World Scientific, 2007.

Church, Alonzo. *The Calculi of Lambda-Conversion.* Princeton University Press, 1941.

Church, Alonzo. *Introduction to Mathematical Logic.* Princeton University Press, 1956.

Clark, Andy and Peter Millican, eds. *Connectionism, Concepts, and Folk Psychology: The Legacy of Alan Turing*, Vol. 2. Clarendon Press, 1996.

Conway, Flo and Jim Siegelman. *Dark Hero of the Information Age: In Search of Norbert Wiener, the Father of Cybernetics.* Basic Books, 2005.

Copeland, B. Jack. *Alan Turing's Automatic Computing Engine.* Oxford University Press, 2005.

Copeland, B. Jack. *The Essential Turing.* Oxford University Press, 2004.

Dauben, Joseph Warren. *Georg Cantor: His Mathematics and Philosophy of the Infinite.* Princeton University Press, 1979.

Davis, Martin. *Computability and Unsolvability.* McGraw-Hill, 1958. Enlarged edition, Dover, 1982.

Davis, Martin. *The Universal Computer: The Road from Leibniz to Turing.* W. W. Norton, 2000. Published in paperback under the title *Engines of Logic: Mathematicians and the Origin of the Computer.*

Davis, Martin, ed. *The Undecidable: Basic Paper on Undecidable Propositions, Unsolvable Problems and Computable Functions.* Raven Press, 1965.

Dawson, John W. Jr. *Logical Dilemmas: The Life and Work of Kurt Gödel.* A. K. Peters, 1997.

DeLong, Howard. *A Profile of Mathematical Logic.* Addison-Wesley, 1970; Dover, 2004.

Dennett, Daniel. *Consciousness Explained.* Back Bay Books, 1991.

Devlin, Keith. *Mathematics: The New Golden Age.* Penguin, 1988.

Enderton, Herbert B. *A Mathematical Introduction to Logic,* second edition. Harcourt, 2001.

Epstein, Richard L. and Walter A. Carnielli. *Computability: Computable*

Functions, Logic, and the Foundations of Mathematics, second edition. Wadsworth, 2000.

Eves, Howard. *Foundations and Fundamental Concepts of Mathematics,* third edition. PWS-Kent, 1990; Dover, 1997.

Ewald, William. *From Kant to Hilbert: A Source Book in the Foundations of Mathematics*, in two volumes. Clarendon Press, 1996.

Franzén, Torkel. *Gödel's Theorem: An Incomplete Guide to Its Use and Abuse.* A. K. Peters, 2005.

Gödel, Kurt. *Collected Works, Volume I: Publications 1929-1936,* edited by Solomon Feferman et. al. Oxford University Press, 1986.

Gödel, Kurt. *Collected Works, Volume II: Publications 1938-1974*, edited by Solomon Feferman et. al. Oxford University Press, 1990.

Gödel, Kurt. *Collected Works, Volume III: Unpublished Essays and Lectures,* edited by Solomon Feferman et. al. Oxford University Press, 1995.

Goldstein, Rebecca. *Incompleteness: The Proof and Paradox of Kurt Gödel.* W. W. Norton, 2005.

Goldstern, Martin and Haim Judah. *The Incompleteness Phenomenon: A New Course in Mathematical Logic.* A. K. Peters, 1995.

Grattan-Guiness, I. *The Search for Mathematical Roots, 1870-1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel.* Princeton University Press, 2000.

Gray, Jack and Keith Thrower. *How the Turing Bombe Smashed the Enigma Code.* Speedwell, 2001.

Heath, Sir Thomas L. *Diophantus of Alexandria: A Study in the History of Greek Algebra,* second edition. Cambridge University Press, 1910; Dover, 1964.

Heims, Steve J. *John von Neumann and Norbert Wiener: From Mathematics to the Technologies of Life and Death.* MIT Press, 1980.

Herkin, Rolf, ed. *The Universal Turing Machine: A Half-Century Introduction.* Oxford University Press, 1988. Second edition, Springer-Verlag, 1995.

Hilbert, David and Wilhelm Ackermann. *Grundzüge der Theoretischen Logik.* Springer, 1928. Second edition, Springer, 1938; Dover, 1946.

Hilbert, David and Wilhelm Ackermann. *Principles of Mathematical Logic* (translation of the second German edition of *Grundzüge der Theoretischen Logik*). Chelsea, 1950.

Hodges, Andrew. *Alan Turing: The Enigma.* Simon & Schuster, 1983.

Hodges, Andrew. *Turing: A Natural Philosopher*. Phoenix, 1997.
Hofstadter, Douglas R. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, 1979.
Hunter, Geoffrey. *Metalogic: An Introduction to the Metatheory of Standard Eirst Order Logic*. University of California Press, 1971.
Kleene, Stephen Cole. *Introduction to Metamathematics.* Van Nostrand, 1952.
Kleene, Stephen Cole. *Mathematical Logic.* John Wiley & Sons, 1967; Dover, 2002.
Kline, Morris. *Mathematics: The Loss of Certainty*. Oxford University Press, 1980.
Kneale, William and Martha Kneale. *The Development of Logic*. Clarendon Press, 1962.
Leavitt, David. *The Man Who Knew Too Much: Alan Turing and the Invention of the Computer*. W. W. Norton, 2006.
Levin, Janna. *A Madman Dreams of Turing Machines.* Alfred A. Knopf, 2006.
Lewis, C. I. *A Survey of Symbolic Logic.* Univesity of California Press, 1918; Dover, 1960.
Mancosu, Paolo. *Erom Brouwer to Hilbert: The Debate on the Foundations of Mathematics in the 1920s.* Oxford University Press, 1998.
Matiyasevich, Yuri M. *Hilbert's Tenth Problem.* MIT Press, 1993.
Mendelson, Elliott. *Introduction to Mathematical Logic,* fourth edition. Chapman & Hall, 1997.
Millican, Peter and Andy Clark, eds. *Machines and Thought: The Legacy of Alan Turing,* Vol. 1. Clarendon Press, 1996.
Niven, Ivan. *Numbers: Rational and Irrational.* Mathematical Association of America, 1961.
Ore, Oystein. *Number Theory and Its History*. McGraw-Hill, 1948; Dover, 1988.
Peano, Giuseppe. *Selected Works of Giuseppe Peano*, translated and edited by Hubert C. Kennedy. George Allen & Unwin, 1973.
Penrose, Roger. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics.* Oxford University Press, 1989.
Petzold, Charles. *Code: The Hidden Language of Computer Hardware and Software.* Microsoft Press, 1999.
Phillips, Esther R., ed. *Studies in the History of Mathematics (MAA Studies in*

Mathematics, Volume 26). Mathematical Association of America, 1987.
Prager, John. *On Turing*. Wadsworth, 2001.
Quine, Willard Van Orman. *Mathematical Logic*, revised edition. Harvard University Press, 1951, 1981.
Reid, Constance. *Hilbert.* Springer-Verlag, 1970, 1996.
Reid, Constance. *Julia: A Life in Mathematics.* Mathematical Association of America, 1996.
Robinson, Julia. *The Collected Works of Julia Robinson,* edited by Solomon Feferman. American Mathematical Society, 1996.
Russell, Bertrand. *Introduction to Mathematical Philosophy,* second edition. George Allen & Unwin, 1920; Dover, 1993.
Russell, Bertrand. *The Principles of Mathematics.* Cambridge University Press, 1903. W. W. Norton, 1938.
Shanker, S. G., ed. *Gödel's Theorem in Focus.* Routledge, 1988.
Sipser, Michael. *Introduction to the Theory of Computation.* PWS, 1997.
Teuscher, Christof, ed. *Alan Turing: Life and Legacy of a Great Thinker*. Springer, 2004.
Tiles, Mary. *The Philosophy of Set Theory: An Historical Introduction to Cantor's Paradise.* Basil Blackwell, 1989; Dover, 2004.
Turing, A. M. *Collected Works of A.* M. *Turing: Mathematical Logic,* edited by R. O. Gandy and C. E. M. Yates. Elsevier, 2001.
Turing, A. M. *Collected Works of A. M. Turing: Mechanical Intelligence,* edited by D. C. Ince. North-Holland, 1992.
Turing, A. M. *Collected Works of A. M. Turing: Morphogenesis,* edited by P. T. Saunders. North-Holland, 1992.
Turing, A. M. *Collected Works of A. M. Turing: Pure Mathematics,* edited by J. L. Britton. North-Holland, 1992.
van Atten, Mark. *On Brouwer*. Wadsworth, 2004.
van Heijenoort, Jean. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931.* Harvard University Press, 1967.
von Neumann, John. *The Computer and the Brain.* Yale University Press, 1958.
Wang, Hao. *Popular Lectures on Mathematical Logic.* Van Nostrand Reinhold, 1981; Dover, 1993.
Wang, Hao. *Reflections on Kurt Gödel.* MIT Press, 1987.
Whitehead, Alfred North and Bertrand Russell. *Principia Mathematics to *56.* Cambridge University Press, 1962.

Whitemore, Hugh. *Breaking the Code.* Fireside Theatre, 1987.
Wiener, Norbert. *Cybernetics, or Control and Communication in the Animal and the Machine.* John Wiley & Sons, 1948. Second edition, MIT Press, 1961.
Wilder, Raymond L. *Introduction to the Foundations of Mathematics.* John Wiley & Sons, 1952.
Wolfram, Stephen. *A New Kind of Science.* Wolfram Media, 2002.
Yates, David M. *Turing's Legacy: A History of Computing at the National Physical Laboratory 1945-1995.* Science Museum (London), 1997.

Table of Contents